IL MANUALE DELL'AMIGADOS

Commodore-Amiga Inc.

IHT
GRUPPO EDITORIALE

IL MANUALE DELL'AMIGADOS

IL MANUALE DELL'AMIGADOS

Commodore-Amiga Inc.

Ringraziamenti

Questo libro è stato originariamente scritto da Tim King e poi completamente revisionato da Jessica King. Un ringraziamento speciale va a Patria Brown, i cui suggerimenti editoriali hanno contribuito in modo sostanziale alla qualità del manuale.

Ringraziamenti vanno anche a Bruce Barrett, Keith Stobie, Robert Peck e a tutti gli altri che alla Commodore-Amiga hanno controllato con cura i contenuti. A Tim King, Paul Floyd e Alan Cosslett della Metacomco va il merito di una preziosa opera editoriale. Infine, va anche riconosciuto il merito di Pamela Clare e Liz Leban per quel che riguarda la revisione.

Sommario

Il Manuale dell'AmigaDOS per l'utente 1

Il Manuale dell'AmigaDOS per il programmatore 197

Il Manuale dell'AmigaDOS di riferimento tecnico 286

Indice analitico 357

Prefazione

Questo libro, *Il Manuale dell'AmigaDOS*, è la combinazione di tre diverse pubblicazioni:

Il Manuale dell'AmigaDOS per l'utente
Il Manuale dell'AmigaDOS per il programmatore
Il Manuale dell'AmigaDOS di riferimento tecnico

Il Manuale per l'utente contiene informazioni che interessano ogni utente dell'Amiga, dal momento che esistono più comandi eseguibili dall'AmigaDOS di quanti siano quelli accessibili dal Workbench. Per servirsi di questi comandi è sufficiente attivare il CLI (Command Line Interface, interfaccia linea comando) tramite il programma Preferences.

Il Manuale per il programmatore descrive la gestione dell'AmigaDOS attraverso un programma piuttosto che tramite l'interfaccia linea comando. Inoltre documenta in modo completo l'Amiga Macro Assembler e il Linker, disponibili come prodotti separati.

Il Manuale di riferimento tecnico descrive le strutture dei dati che l'AmigaDOS adotta internamente. Include la spiegazione di come i dati sono memorizzati in un disco DOS e la descrizione del formato dei "file oggetto" usato dall'AmigaDOS. Il programmatore e l'utente esperto troveranno estremamente utili le informazioni contenute in questa sezione tecnica.

L'unione di queste tre pubblicazioni in un singolo volume rappresenta la miglior guida all'AmigaDOS che si possa desiderare.

Il Manuale dell'AmigaDOS per l'utente

Indice

1. Introduzione all'AmigaDOS 3
2. I comandi dell'AmigaDOS 43
3. ED - L'editor di schermo. 130
4. EDIT - L'editor di linea. 146

Appendice: codici e messaggi d'errore. 189
Glossario 194

Capitolo 1

Introduzione all'AmigaDOS

Questo capitolo fornisce una descrizione generale del sistema operativo AmigaDOS. Include una descrizione della gestione del terminale, la struttura delle directory e l'uso dei comandi. Alla fine del capitolo è presentato un esempio di una semplice sessione di lavoro con l'AmigaDOS.

1.1 Descrizione del capitolo
1.2 Gestione del terminale
1.3 Uso del filing system
1.3.1 Nomi dei file
1.3.2 Uso delle directory
1.3.3 Come impostare la directory corrente
1.3.4 Selezione del device corrente
1.3.5 Aggiungere una nota ai file
1.3.6 Comprensione dei nomi dei device
1.3.7 Uso delle convenzioni per le directory e impiego dei device logici
1.4 Uso dei comandi dell'AmigaDOS
1.4.1 Esecuzione dei comandi in background
1.4.2 Esecuzione dei file di comandi
1.4.3 Ridirezione dell'input e dell'output dei comandi
1.4.4 Interruzione dell'AmigaDOS
1.4.5 Comprensione dei formati dei comandi
1.5 Processo di convalida
1.6 Comandi d'uso più comune
1.7 Convenzioni adottate

1.1 Descrizione del capitolo

L'AmigaDOS è un sistema operativo **multitasking**, progettato per l'Amiga. Sebbene possa essere usato come sistema multi-utente, l'AmigaDOS viene normalmente utilizzato da un utente singolo. La disponibilità del

multitasking permette l'esecuzione simultanea di processi diversi. Si può anche usare il multitasking per sospendere un lavoro mentre ne viene eseguito un altro.

Ogni **processo** (task) dell'AmigaDOS, come ad esempio il filing system, rappresenta un particolare task del sistema operativo. Viene eseguito solamente un processo alla volta; quelli rimanenti correntemente attivi possono trovarsi in attesa che si verifichi una particolare condizione, o sono stati sospesi e attendono di poter riprendere la loro esecuzione. A ogni processo è associata una **priorità**, e quello che ha la priorità più elevata è libero di procedere per conto proprio. Gli altri vengono eseguiti solo quando quelli con priorità più alta sono per qualche ragione in attesa di qualcosa; ad esempio di ricevere delle informazioni dal disco.

Il sistema AmigaDOS standard si serve di un certo numero di task che non sono disponibili all'utente, come per esempio, quello che controlla la porta seriale. Questi processi sono conosciuti come "processi riservati". Altri task riservati sono devoluti al controllo del terminale e al filing system di un disk drive. Se la configurazione hardware contiene più di un disk drive, è presente un processo per ogni drive.

L'AmigaDOS fornisce un processo utilizzabile dall'utente, chiamato **Interfaccia Linea Comando** o **CLI**. Possono essere eseguiti simultaneamente diversi processi CLI, numerati progressivamente a partire da 1. I processi CLI leggono i **comandi** e li eseguono. Tutti i comandi e i programmi degli utenti possono essere eseguiti da qualsiasi CLI. Per creare processi CLI addizionali si possono usare i comandi NEWCLI e RUN. Per rimuovere un processo CLI si usa invece il comando ENDCLI. Nel capitolo 2 si trova la descrizione completa di questi comandi.

1.2 Gestione del terminale

Le informazioni che vengono introdotte nel terminale hanno due strade da seguire: possono dirigersi verso un CLI che può comunicare all'AmigaDOS di caricare un programma, oppure possono venire inoltrate a un programma che viene eseguito sotto quel dato CLI. In entrambi i casi, è un **gestore del terminale** (o **console**) che elabora sia l'input che l'output. Inoltre tale processo compie il trattamento locale della linea di testo e alcune altre funzioni. La lunghezza massima di una linea digitata è di 255 caratteri.

Per correggere gli errori, occorre premere il tasto BACKSPACE, che cancella l'ultimo carattere digitato. Per cancellare un'intera riga, si può tenere premuto il tasto CTRL contemporaneamente al tasto X. Questo tipo di combinazioni con il tasto CTRL, chiamate anche **combinazioni control**, vengono d'ora in poi indicate in questo manuale come CTRL-X.

Se state scrivendo qualcosa, l'AmigaDOS attende finché non avete terminato l'operazione prima di visualizzare qualunque altro output. Potete

così scrivere senza che l'input e l'output vengano mischiati. L'AmigaDOS riconosce che avete terminato l'introduzione di una linea di testo quando premete il tasto RETURN. Si può anche comunicare all'AmigaDOS di aver terminato una linea cancellandola per intero. Per cancellare una linea si può premere CTRL-X oppure una serie di BACKSPACE finché tutti i caratteri della linea non sono stati cancellati. Quando l'AmigaDOS si rende conto che avete terminato, riprende la visualizzazione dell'output che aveva tenuto in sospeso. Se desiderate fermare l'output per poterlo leggere, potete semplicemente digitare un qualunque carattere (premere la barra spaziatrice è la cosa più semplice) e l'output si ferma. Per riprendere l'output premete BACKSPACE, CTRL-X o RETURN. La pressione di RETURN fa in modo che l'AmigaDOS tenti di eseguire la linea digitata solo dopo il termine del programma corrente.

L'AmigaDOS riconosce CTRL-\ come indicatore di fine file. In certe circostanze, si può utilizzare questa combinazione per chiudere un file di input (vedere la sezione 1.3.6 per determinare quando si può usare CTRL-\ ).

Se vedete che sullo schermo appaiono strani caratteri quando scrivete qualcosa sulla tastiera, probabilmente avete premuto CTRL-O inavvertitamente. L'AmigaDOS riconosce questa combinazione come istruzione al console device (CON:) per visualizzare il set di caratteri alternativo. Per tornare alla situazione iniziale premete CTRL-N. Qualunque altro carattere apparirà ora normalmente. D'altro canto potreste premere ESC-C per pulire lo schermo e visualizzare il testo normale.

Nota: *CON:, il device di controllo della tastiera, non prende in considerazione la pressione dei tasti funzione e di quelli per lo spostamento del cursore. Se desiderate che questi tasti vengano riconosciuti dovete utilizzare RAW:. Per una descrizione di RAW:, vedere la sezione 1.3.6, "Comprensione dei nomi dei device", più avanti in questo stesso capitolo.*

Infine l'AmigaDOS riconosce tutti i comandi e gli **argomenti** sia in maiuscolo che in minuscolo. L'AmigaDOS visualizza un **nome di file** con i caratteri, maiuscoli o minuscoli, che sono stati usati quando il file è stato creato, ma non ha nessuna importanza il tipo di carattere che usate quando fate riferimento al suo nome per identificarlo.

1.3 Uso del filing system

Questa sezione descrive l'uso del filing system dell'AmigaDOS. In particolare spiega come attribuire un nome ai file, organizzarli e richiamarli.

Un file è il più piccolo oggetto dotato di nome usato dall'AmigaDOS. Il modo più semplice per identificare un file è attraverso il suo nome, come illustra qui di seguito la sezione 1.3.1. Comunque, può rendersi necessario identificare un file in un modo più completo. Un'identificazione di questo tipo potrebbe includere il nome del device o del volume, il nome della directory

così come il nome del file. Questo argomento verrà illustrato nelle sezioni seguenti.

1.3.1 Nomi dei file

L' AmigaDOS mantiene le informazioni su disco in una serie di **file**, i quali sono dotati di un nome, in modo da poter essere identificati e richiamati. Il filing system permette che i nomi dei file siano lunghi fino a trenta caratteri, e i caratteri siano tutti quelli stampabili eccetto la barra (/) e i due punti (:). Questo significa che all'interno di un nome di file si possono includere spazi ( ), segni di uguale (=), segni di addizione (+) , le virgolette e tutti i caratteri speciali riconosciuti dal CLI. In questo particolare caso però, si deve avere l'accortezza di racchiudere tra virgolette il nome del file ed eventualmente l'intero percorso (vedere i paragrafi seguenti). Per introdurre le virgolette nel nome di un file, bisogna scrivere un asterisco (*) immediatamente prima di questo carattere. Inoltre, per introdurre un asterisco, bisogna scriverne un altro. Questo significa che un file dal nome

```
A*B = C''
```

deve essere scritto come segue:

```
''A**B = C*''''
```

perché il CLI lo accetti.

Nota: *quest'uso dell'asterisco contrasta con quello della maggior parte dei sistemi operativi, nei quali viene impiegato come* **wild card** *universale. Un asterisco da solo rappresenta per l'AmigaDOS la tastiera e la finestra corrente. Per esempio,*

```
COPY nomedelfile TO *
```

copia il contenuto del file specificato sullo schermo.

1.3.2 Uso delle directory

Il filing system permette anche l'uso delle **directory** per raggruppare i file in unità logiche. Per esempio, si possono usare due directory differenti per separare i programmi sorgenti dalle loro documentazioni, oppure per mantenere i file che appartengono a una persona separati da quelli che appartengono a un'altra.

Ogni file deve appartenere a una directory. Un disco vuoto contiene una directory, chiamata **directory radice** (root directory). Se viene creato un file in un disco vuoto, il sistema lo inserisce nella directory radice. Comunque, le directory stesse possono contenere ulteriori directory. Ogni directory può quindi contenere file, oppure altre directory, o un misto di entrambi. Qualunque nome di file è unico solamente all'interno della directory cui appartiene, così il file "alberto" della directory "marco" è un file completamente differente da quello chiamato "alberto" nella directory "roberta".

Questa struttura dei file permette che due persone che condividono lo stesso disco non debbano preoccuparsi della sovrascrittura di file creati da qualcun altro, finché creano i propri file nelle proprie directory.

ATTENZIONE: se create un file con un nome che esiste già, l'AmigaDOS cancella il contenuto precedente di quel file. Sullo schermo non troverete nessun messaggio che segnali questa operazione.

Si può usare questa struttura a directory anche per organizzare le informazioni che esistono sul disco, tenendo diversi tipi di file in directory differenti.

Un esempio può aiutarvi a chiarire questo aspetto. Consideriamo un disco che contiene due directory chiamate "marco" e "roberta". La directory "marco" contiene due file chiamati "testo" e "lettera". La directory "roberta" contiene un file chiamato "dati" e due directory chiamate "lettera" e "fattura". Queste subdirectory contengono ognuna un file chiamato "18giugno". La Tavola 1-A rappresenta questa struttura:

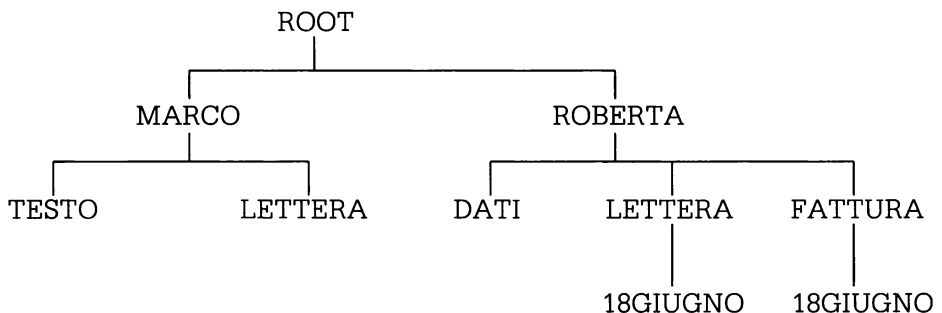



Tavola 1-A: una struttura di directory

Nota: *la directory "marco" ha un file chiamato "lettera", mentre la directory "roberta" ne contiene un'altra chiamata "lettera". Non si crea comunque confusione, perché i due file sono in directory differenti. Non c'è limite al livello di nidificazione delle directory.*

Per specificare un file in modo completo bisogna includere la directory che lo contiene, la directory che contiene questa directory, e così via, cioè occorre indicare il nome di tutte le directory che si trovano sul percorso per arrivare al file desiderato. Per separare ogni nome di directory dalla successiva directory o dal nome di file, bisogna introdurre una barra (/). Così, le indicazioni complete dei file presenti nel disco della Tavola 1-A sono le seguenti:

```
marco/testo
marco/lettera
roberta/dati
roberta/lettera/18giugno
roberta/fattura/18giugno
```

1.3.3 Come impostare la directory corrente

L'indicazione completa di un file può essere estremamente scomoda da digitare, così il filing system crea e mantiene una **directory corrente** fittizia dove vengono ricercati i file. Per specificare la directory corrente bisogna usare il comando CD (dall'inglese Current Directory). Se avete selezionato "roberta" come vostra directory corrente allora i nomi che seguono sono sufficienti per specificare i file in essa contenuti:

```
dati
lettera/18giugno
fattura/18giugno
```

Si può impostare come directory corrente una qualunque delle directory presenti su disco. Per indicare un file al suo interno basta semplicemente scriverne il nome. Per specificare i file all'interno delle subdirectory, bisogna digitare i nomi delle directory poste lungo il percorso a partire dalla directory corrente.

Tutti i file residenti su un disco restano disponibili anche se è stata impostata una directory corrente. Per far sì che l'AmigaDOS li ricerchi attraverso tutte le directory a partire dalla radice, è necessario aggiungere i due punti (:) all'inizio della descrizione del file. Così, quando avete impostato "roberta" come directory corrente, potete ottenere il file "dati" anche inserendo la descrizione ":roberta/dati". L'uso della directory corrente riduce solamente la battitura, visto che tutto ciò che si deve fare è indicare il nome del file "dati".

Per ottenere gli altri file presenti sul disco, digitate rispettivamente

":marco/testo" e ":marco/lettera". Un altro metodo può essere quello di usare il comando CD o di aggiungere / prima del nome di un file. La barra non significa "radice" come in alcuni sistemi, ma si riferisce alla directory precedente alla directory corrente. L'AmigaDOS permette l'uso multiplo di barre, con ogni barra che si riferisce a un livello precedente. Così un ../ in Unix equivale a / in AmigaDOS. Similmente, un ..\ in MS-DOS equivale a / in AmigaDOS. Così, se la directory corrente è ":roberta/lettera", potete specificare il file ":roberta/fattura/18giugno" come "/fattura/18giugno". Per riferirsi ai file in ":marco", potete scrivere:

```
CD :marco
```

definendo direttamente la nuova directory corrente, oppure

```
CD //marco
```

definendo la nuova directory corrente a partire dalla posizione precedente, cioè "roberta/lettera". Il primo / fa risalire a "roberta", il secondo alla directory radice. In seguito potete specificare qualunque file in "marco" con un singolo nome di file. Naturalmente potete sempre usare // per riferirvi direttamente a uno specifico file. Per esempio,

```
TYPE //marco/lettera
```

visualizza il file senza che abbiate impostato "marco" come directory corrente. Per andare subito al livello della radice, conviene usare sempre i due punti (:) seguiti dal nome di una directory. Se si usano le barre, bisogna conoscere esattamente il numero di livelli da percorrere a ritroso.

1.3.4 Selezione del device corrente

È possibile che l'utente disponga di più disk drive. A ognuna di queste periferiche viene assegnato un nome, nella forma dfn: (per esempio, df1:), dove "n" si riferisce al numero del device (attualmente l'AmigaDOS accetta i nomi di device da df0: a df3:). Ogni singolo disco è assooiato a un nome unico, conosciuto come nome del volume (vedere più avanti per ulteriori dettagli).

In aggiunta a ciò, il device logico SYS: viene assegnato al disco dal quale fate partire il sistema. Si può usare questo nome al posto del nome del disco (come df0:).

La directory corrente è associata anche a un **device corrente**, cioè il drive nel quale risiede tale directory. È noto che porre i due punti prima della descrizione di un file serve per identificare la directory radice del drive

corrente. Se invece si desidera identificare la directory radice di un particolare drive, per esempio diverso da quello corrente, si deve collocare prima dei due punti il nome del drive. Così, supponendo che abbiate inserito il disco nel drive df1, avete un altro modo per riferirvi al file "dati" nella directory "roberta", cioè "df1:roberta/dati". Per riferirvi a un file nel drive df0 chiamato "progetto" appartenente alla directory "pietro", potreste immettere "df0:pietro/progetto", senza preoccuparvi di quale directory sia impostata come quella corrente.

Nota: *quando ci si riferisce a un disk drive o a un qualunque altro device da solo o in unione a un nome di directory, bisogna sempre aggiungere i due punti, per esempio df1:.*

La Tavola 1-B illustra la struttura di una descrizione di file. La Tavola 1-C presenta alcuni esempi di corrette descrizioni di file.

A sinistra dei ":"	**A destra dei ":"**	**A destra di "/"**
Nome del device	Nome della directory	Nome della subdirectory
oppure	oppure	oppure
Nome del volume	Nome del file	Nome del file

Tavola 1-B: la struttura di una descrizione di file

```
SYS:comandi
DF0:marco
DF1:roberta/lettera
DF2:roberta/lettera/18giugno
DOC:progetto/sezione1/figure
FONTS:font-prova
C:cls
```

Tavola 1-C: esempi di descrizioni di file

Per avere accesso a un file presente in un disco particolare, si può scrivere il nome unico del disco, che è conosciuto come il **nome di volume** del disco, al posto del nome del device. Per esempio se il file è sul disco "MCC", potete specificarlo indicando il nome "MCC:pietro/progetto". Si può usare il nome di volume per riferirsi a un disco senza indicare in quale drive si trovi. Il nome di volume viene assegnato a un disco quando lo si formatta (per ulteriori dettagli, vedere il comando FORMAT nel capitolo 2 di questo manuale).

Un nome di device, a differenza di un nome di volume, non fa realmente parte del nome. Per esempio, l'AmigaDOS può leggere un file creato in df0: da un altro drive, come df1:, se inserite il disco in quel drive, evidenziando

di conseguenza l'intercambiabilità dei drive. Se create un file "marco" sul disco nel drive df0:, il file è identificato come "df0:marco". Se spostate il disco nel drive df1:, l'AmigaDOS può ancora leggere il file, che ora però è identificabile come "df1:marco".

1.3.5 Aggiungere una nota ai file

Sebbene il nome di un file possa evidenziare alcune informazioni sul suo contenuto, si rende spesso necessario guardare all'interno del file stesso per ottenere maggiori dettagli. L'AmigaDOS fornisce una semplice soluzione a questo problema. Si può usare il comando FILENOTE per aggiungere al file un commento lungo fino a 80 caratteri (ricordandosi di racchiudere tra virgolette i commenti che comprendono degli spazi). Nel commento di un file si può inserire qualunque informazione: il giorno della creazione del file, se un errore è stato corretto oppure no, la versione del programma e qualunque altra cosa possa aiutare la sua identificazione.

Il commento deve essere associato a un file particolare (non è necessario che tutti abbiano dei commenti). Per aggiungere il commento occorre impartire il comando FILENOTE. Un file creato ex novo non è dotato di commento. Quando si copia un file dotato di commento in un nuovo file, viene trasferito il contenuto ma non il commento. Comunque, ogni commento aggiunto a un file che viene riscritto più volte viene mantenuto. Si può scrivere un programma che copi un file e il suo commento, ma occorre fare uno specifico lavoro supplementare per copiare il commento. Per ulteriori particolari su questo argomento, vedere il capitolo 2 del *Manuale dell'AmigaDOS per il programmatore*.

Quando si cambia nome a un file, il commento associato non cambia, visto che il comando RENAME interviene solo sul nome del file. Il contenuto del file e il suo commento rimangono inalterati, comunque se ne alteri il nome. Per ulteriori particolari, riferirsi ai comandi LIST e FILENOTE illustrati nel capitolo 2 di questo manuale.

1.3.6 Comprensione dei nomi dei device

I device sono dotati di un nome, in modo che sia possibile riferirsi a loro usando un termine preciso. I nomi dei dischi, come df0:, sono esempi di **nomi di device**. Notate che per riferirsi a un nome di device, così come per un nome di file, si possono usare sia caratteri maiuscoli che minuscoli. Per quanto riguarda i dischi, si fa seguire al nome di device un nome di file poiché l'AmigaDOS supporta i file in questi device. Inoltre un nome di file può includere delle directory, visto che l'AmigaDOS supporta anche quelle.

Si possono creare anche file in memoria utilizzando il device chiamato

RAM:. Questo dispositivo emula un disk drive in memoria; è quindi in grado di creare un filing system che utilizza qualunque comando del filing system normale.

Nota: *il device RAM: richiede la presenza del file "l/ram-handler" sul disco. Nella versione 1.2, il device RAM: scrive i file in blocchi di 512 byte. Per attivare l'icona del device RAM: ogni volta che viene fatto partire il sistema, bisogna modificare il file di startup per includere il comando appropriato.*

Una volta che il RAM: esiste, si può, per esempio, creare una directory per copiare tutti i comandi in memoria, usando le seguenti istruzioni:

```
MAKEDIR ram:c
COPY sys:c TO ram:c
ASSIGN C: RAM:C
```

Ora potete vedere il risultato con DIR RAM:. Sarà presente la directory "c", che il comando DIR visualizza come c (dir). Questa operazione renderà l'esecuzione dei comandi molto veloce ma lascerà disponibile poca memoria per altre applicazioni. Ogni file presente nel device RAM: viene perso quando la macchina subisce un reset.

Nota: *sul disco Workbench non tutti i comandi dell'AmigaDOS sono contenuti nella directory c. Quindi copiando tale directory nel drive in memoria RAM: e assegnandola, può accadere che impartendo determinati comandi il sistema, accedendo esclusivamente al dispositivo RAM:, non li trovi e non possa quindi mandarli in esecuzione. Per esempio il comando FORMAT è residente nella directory System, e non verrebbe quindi eseguito.*

L'AmigaDOS fornisce altri device che potete usare senza bisogno di far riferimento a un file su disco. I paragrafi seguenti descrivono tali device, tra i quali NIL:, SER:, PAR:, PRT:, CON: e RAW:. In particolare il device NIL: è un device fittizio. L'AmigaDOS non crea nessun output diretto a NIL:, mentre, quando si legge da NIL:, l'AmigaDOS fornisce immediatamente un'indicazione di "end-of-file" (fine del file). Per esempio, potete scrivere:

```
EDIT abc TO nil:
```

in modo da usare l' editor per scorrere un file, senza che l'AmigaDOS riscriva le eventuali modifiche.

Si può usare il device chiamato SER: per riferirsi a qualunque periferica connessa tramite la linea seriale (spesso una stampante). Così, potete scrivere il comando seguente:

```
COPY xyz TO ser:
```

per comunicare all'AmigaDOS di inviare il contenuto del file "xyz" alla linea seriale. Bisogna notare che il device seriale copia soltanto in multipli di 400 byte alla volta; quindi effettuare una copia con SER: può apparire discontinuo.

Il device PAR: si riferisce alla porta parallela nello stesso modo.

L'AmigaDOS fornisce inoltre il device PRT: (dall'inglese PRinTer, che significa stampante). Il device PRT: identifica la stampante che utilizzate, definita per mezzo del programma "Preferences". In questo programma, potete definire se la vostra stampante è collegata attraverso la porta seriale o quella parallela. Così, il comando

```
COPY xyz TO prt:
```

stampa il file "xyz" prescindendo dalla porta alla quale è connessa la stampante.

Il PRT: trasforma ogni carattere di line feed (avanzamento linea), presente in un file, in un carriage return (ritorno carrello) seguito dal carattere line feed. Alcune stampanti, però, richiedono i file senza trasformazione. Per trasmettere un file senza che i linefeed vengano alterati, occorre usare PRT:RAW al posto di PRT:.

L'AmigaDOS consente l'uso di finestre multiple. Per formare una nuova finestra, si può usare il device CON:. Il formato per il device CON: è:

```
CON:x/y/larghezza/altezza/[titolo]
```

dove "x" e "y" sono le coordinate, "larghezza" e "altezza" sono numeri interi rappresentanti la larghezza e l'altezza della nuova finestra, e "titolo", che è facoltativo, è una stringa. Bisogna includere tutte le barre (/), compresa l'ultima. Il titolo, che appare nell'intestazione della finestra, può essere composto da un massimo di trenta caratteri (spazi inclusi). Se il titolo contiene degli spazi bisogna racchiudere l'intera descrizione tra virgolette ('') come si vede nell' esempio seguente:

```
''CON:20/10/300/100/prova finestra''
```

C'è un altro device che riguarda le finestre chiamato RAW:, ma viene impiegato raramente dall'utente medio (vedere il capitolo 2 del *Manuale dell'AmigaDOS per il programmatore* per ulteriori particolari). Si può usare RAW:, nello stesso modo in cui si usa CON:, per creare un device che tratta le finestre in modo meno elegante. RAW: però, al contrario di CON:, non effettua la traduzione dei caratteri e non permette di modificare il contenuto di una linea, cioè accetta l'input e restituisce l'output esattamente nella stessa forma in cui sono stati originariamente scritti. Questo significa che i caratteri sono mandati immediatamente a un programma senza permettere

di cancellarne qualcuno usando il tasto BACKSPACE. Normalmente si usa RAW: con un programma che pretende l'input e l'output senza la traduzione dei caratteri.

ATTENZIONE: il device RAW: deve essere utilizzato solo dai più esperti. È consigliabile servirsene con cautela.

Per riferirsi alla finestra corrente, sia per l'input che per l'output, è possibile indicarla con un nome speciale: * (il carattere asterisco). Il comando COPY può essere usato per copiare da un file a un altro; quindi usando *, è possibile copiare il contenuto di una finestra in un'altra. Per esempio:

```
COPY * TO CON:20/20/350/150/
```

oppure dalla finestra corrente alla finestra corrente. Per esempio:

```
COPY * TO *
```

oppure da un file alla finestra corrente. Per esempio:

```
COPY marco/lettera TO *
```

L'AmigaDOS finisce di copiare quando giunge alla fine del file. Per comunicare all'AmigaDOS di terminare la copia dal *, bisogna impartire la combinazione CTRL-\. Notate che * NON è la wild card (cioè il carattere jolly) universale.

1.3.7 Uso delle convenzioni per le directory e impiego dei device logici

Oltre ai device fisici che abbiamo già menzionato, l'AmigaDOS supporta una serie di utili **device logici**. L'AmigaDOS usa tali device per individuare i file che i programmi richiedono di volta in volta. In questo modo i programmi possono riferirsi a un nome di device standard prescindendo dalla reale locazione del file. Tutti questi device logici possono venire riassegnati dall'utente per riferirsi a qualunque directory.

I device logici descritti in questa sezione sono elencati nella tavola della pagina successiva.

Nome	Descrizione	Directory
SYS:	Directory radice del disco di sistema	:
C:	Directory dei comandi	:C
L:	Directory delle librerie	:L
S:	Libreria delle sequenze	:S
LIBS:	Libreria per le chiamate a OpenLibrary	:LIBS
DEVS:	Device per le chiamate a OpenDevice	:DEVS
FONTS:	Font caricabili per le chiamate a OpenDiskFont	:FONTS
T:	Area di lavoro temporanea da assegnare	:T

Tavola 1-D: i device logici

Device logico: SYS:

Directory tipica: Disco.Iniziale:

Descrizione: "SYS" rappresenta la directory radice del SYStem disk (disco sistema). Quando si attiva l'Amiga, l'AmigaDOS assegna il device logico SYS: al nome della directory radice del disco presente in df0:. Se, per esempio, il disco nel drive df0: è dotato del nome di volume "Disco.Iniziale", l'AmigaDOS assegna SYS: a Disco.Iniziale:. Dopo questa assegnazione, qualunque programma che si riferisca a SYS: userà la directory radice di quel disco.

Device logico: C:

Directory tipica: Disco.Iniziale:c

Descrizione: "C" rappresenta la directory dei comandi. Quando si immette un comando da CLI (per esempio, DIR <cr>), l'AmigaDOS inizialmente cerca di individuarlo nella directory corrente. Se il sistema non riesce a trovare il comando nella directory corrente, procede a cercarlo come "C:DIR". Così, se avete assegnato "C:" a un'altra directory (per esempio, "Altro.Disco:c"), l'AmigaDOS legge e quindi esegue "Altro.Disco:c/DIR".

Device logico: L:

Directory tipica: Disco.Iniziale:l

Descrizione: "L" rappresenta la directory delle Librerie. Questa directory contiene gli overlay per i comandi troppo lunghi

e le parti non residenti del sistema operativo. Per esempio, le librerie di run-time mantenute su disco (Ram-Handler, Port-Handler, Disk-Validator...) trovano posto in questa directory. L'AmigaDOS richiede la presenza di questa directory per operare.

Device logico: S:

Directory tipica: Disco.Iniziale:s

Descrizione: "S" rappresenta la libreria delle Sequenze. I file sequenza contengono una serie di comandi che il comando "EXECUTE" cerca ed esegue. EXECUTE ricerca il file sequenza (chiamato anche file batch) inizialmente nella directory corrente. Se l'operazione non ha successo, EXECUTE procede a cercarlo nella directory alla quale è stato assegnato il device logico S:.

Device logico: LIBS:

Directory tipica: Disco.Iniziale:libs

Descrizione: La funzione OpenLibrary (che fa parte del ROM Kernel) ricerca in questa directory le librerie che non sono già presenti in memoria.

Device logico: DEVS:

Directory tipica: Disco.Iniziale:devs

Descrizione: La funzione OpenDevice ricerca in questa directory i device che non sono già presenti in memoria.

Device logico: FONTS:

Directory tipica: Disco.Iniziale:fonts

Descrizione: La funzione OpenDiskFont ricerca in questa directory le fonti carattere caricabili che non sono già presenti in memoria.

Nota: *in aggiunta alle directory assegnabili, appena citate, alcuni programmi aprono dei file nella ":T" directory. Come ricorderete, i file (o le directory) preceduti dai ":" trovano posto nella directory radice. Così ":T"*

individua la directory T, contenuta nella radice, del disco corrente. Si può usare questa directory per memorizzare file temporanei. Gli editor, e in genere i programmi dello stesso tipo, inseriscono i loro file di lavoro temporanei, o le copie di backup dell'ultimo file modificato, in questa directory. Se non avete più spazio a disposizione sul disco, questo è uno dei primi posti dove cercare i file che non servono più.

Quando il sistema viene inizializzato, l'AmigaDOS assegna C: alla directory :C. Questo significa che se si attiva il sistema con un disco che è stato formattato con il comando:

```
FORMAT DRIVE DF0: NAME ''Disco.Iniziale''
```

SYS: viene assegnato a "Disco.Iniziale". Il device logico C: è assegnato alla directory C dello stesso disco (cioè Disco.Iniziale:c). Nello stesso modo, vengono assegnati i seguenti nomi:

```
C:          Disco.Iniziale:c
L:          Disco.Iniziale:l
S:          Disco.Iniziale:s
LIBS:       Disco.Iniziale:libs
DEVS:       Disco.Iniziale:devs
FONTS:      Disco.Iniziale:fonts
```

Se una directory non è presente, il device logico corrispondente viene assegnato alla directory radice.

Se possedete un hard disk (chiamato dh0:) e desiderate usare i file di sistema dovete impartire al sistema i seguenti comandi:

```
ASSIGN SYS:     DH0:
ASSIGN C:       DH0:C
ASSIGN L:       DH0:L
ASSIGN S:       DH0:S
ASSIGN LIBS:    DH0:LIBS
ASSIGN DEVS:    DH0:DEVS
ASSIGN FONTS:   DH0:FONTS
```

Bisogna sempre tenere presente che le assegnazioni coinvolgono tutti i processi CLI. Cambiare un'assegnazione all'interno di una finestra significa ripercuotere la variazione in tutte le altre.

Se desiderate usare la vostra libreria speciale di fonti carattere, digitate

```
ASSIGN FONTS: ''Disco Fonti Speciali:mie.fonti''
```

Se desiderate che i vostri comandi vengano eseguiti più velocemente e avete memoria in abbondanza, digitate

```
MAKEDIR ram:c
COPY sys:c ram:c all
ASSIGN C: ram:c
```

Questa operazione copia tutti i normali comandi dell'AmigaDOS nel disco RAM e riassegna la directory dei comandi in modo che il sistema li trovi nel disco virtuale in memoria.

1.4 Uso dei comandi dell'AmigaDOS

Un comando dell'AmigaDOS è costituito dal nome del comando stesso e dai suoi argomenti, se sono previsti. Per eseguire un comando dell'Amiga-DOS bisogna scriverne il nome e gli argomenti dopo il prompt del CLI.

Quando si scrive il nome di un comando, questo viene eseguito come parte della Command Line Interface (CLI). Immediatamente dopo si possono inserire altri comandi, ma l'AmigaDOS non li esegue finché il comando corrente non è terminato. Quando un comando è terminato, appare il prompt corrente del CLI, cioè il cursore con il messaggio impostato dal sistema. In questo caso si dice che il comando è eseguito in modo interattivo.

Il prompt del CLI è inizialmente n>, dove n è il numero del processo CLI. Il messaggio del cursore può comunque essere cambiato per mezzo del comando PROMPT (per ulteriori dettagli riguardo il comando PROMPT, vedere il capitolo 2 di questo manuale.)

ATTENZIONE: se state eseguendo un comando in modo interattivo e per qualche motivo questo fallisce, l'AmigaDOS continua in ogni caso a eseguire il comando successivo. Perciò, può essere pericoloso introdurre molti comandi in anticipo. Per esempio, se scrivete

```
COPY a TO b
DELETE a
```

e il comando copia fallisce (magari perché il disco è pieno), DELETE viene comunque eseguito e perdete il vostro file.

1.4.1 Esecuzione dei comandi in background

Usando il comando RUN si può ordinare all'AmigaDOS di eseguire uno o più comandi in background (sottofondo). RUN crea un nuovo CLI in qualità

di processo separato a bassa priorità. Se si fa uso di questa possibilità, l'AmigaDOS esegue le linee di comando impostate successivamente, nello stesso momento di quelle che hanno beneficiato dell'uso di RUN. Per esempio si può esaminare il contenuto di una directory mentre un file di testo viene stampato. Per farlo, scrivete

```
RUN TYPE file_di_testo TO prt:
LIST
```

Il comando RUN crea un nuovo CLI e porta avanti la stampa del file, mentre la lista viene visualizzata nella finestra originale del CLI.

Si può chiedere all'AmigaDOS di portare avanti diversi comandi usando RUN. RUN preleva ogni comando e lo esegue nell'ordine dato. La linea contenente i comandi dopo il RUN viene detta command line. Per terminare una command line, bisogna premere RETURN. Per estendere la command line su più linee di testo, si deve aggiungere un simbolo di addizione (+) in ogni linea, eccetto l'ultima, prima di premere RETURN. Per esempio,

```
RUN JOIN file1 file2 AS file3 +
SORT file3 TO file4 +
TYPE file4 TO prt:
```

1.4.2 Esecuzione dei file di comandi

Si può usare anche il comando EXECUTE per eseguire le command line contenute in un file invece che scriverle direttamente. Il CLI legge la sequenza di comandi dal file finché non trova un errore o raggiunge la fine del file. Se trova un errore, l'AmigaDOS non esegue i comandi seguenti sulla linea del RUN o nel file usato da EXECUTE, a meno che non sia stato impartito il comando FAILAT. Per i particolari che riguardano il comando FAILAT, vi rimandiamo al capitolo 2. Il CLI mostra i prompt solo dopo l'esecuzione dei comandi in modo interattivo.

1.4.3 Ridirezione dell'input e dell'output dei comandi

L'AmigaDOS fornisce un modo per ridirigere l'input e l'output standard. Si usano i simboli > e < come comandi. Normalmente, quando si digita un comando, l'AmigaDOS ne visualizza l'output sullo schermo. Per comunicare all'AmigaDOS di mandare l'output a un file si deve usare il comando >. Per accettare l'input verso un programma proveniente da un file invece che dalla tastiera, si deve usare il comando <. I comandi < e > agiscono come vigili addetti al traffico e dirigono il flusso delle informazioni. Per esempio, per dirigere verso il file chiamato "file_di_testo" l'output che deriva dal comando

DATE , dovete digitare la seguente linea di comando:

```
DATE > file_di_testo
```

Consultate il capitolo 2 del *Manuale dell'AmigaDOS per l'utente* per una completa descrizione dei simboli < e >.

1.4.4 Interruzione dell'AmigaDOS

L'AmigaDOS consente quattro livelli diversi di interruzioni con CTRL-C, CTRL-D, CTRL-E e CTRL-F. Per arrestare il comando in corso, qualunque cosa stia facendo, potete premere CTRL-C. In alcuni casi, come in quello di EDIT, la pressione di CTRL-C ordina al comando di fermare ciò che stava facendo, e di tornare alla lettura di ulteriori comandi EDIT. Per comunicare al CLI di fermare una sequenza di comandi iniziata da EXECUTE, si preme CTRL-D appena termina il comando attualmente in esecuzione. CTRL-E e CTRL-F sono usati solamente da certi comandi in casi specifici. Per ulteriori particolari, consultate il *Manuale dell'AmigaDOS per il programmatore* in questo stesso volume.

Nota: *è compito del programmatore riconoscere questi segnali di interruzione e rispondere a essi. L'AmigaDOS, da solo, non interrompe un programma.*

1.4.5 Comprensione dei formati dei comandi

Questa sezione spiega il formato standard o il template (schema) degli argomenti usato dalla maggior parte dei comandi dell'AmigaDOS per specificare i propri argomenti. Il capitolo 2 di questo manuale include il template nella documentazione di ogni comando. Il template fornisce un alto grado di flessibilità per quanto riguarda l'ordine e la forma della sintassi dei comandi.

Il template degli argomenti indica una lista di **keyword** (parole chiave) che possono essere usate come sinonimi. Per farlo, si digita dopo la keyword l'alternativa prescelta, e la si separa con un segno di uguale (=).

Per esempio:

```
ABC,WWW,XYZ = ZZZ
```

specifica le keyword ABC, WWW e XYZ. L'utente può usare la keyword ZZZ al posto della keyword XYZ.

Queste keyword specificano il numero e la forma degli argomenti che il programma si attende. Gli argomenti possono essere opzionali o obbligatori.

Gli argomenti, se vengono forniti, possono essere specificati in due modi:

Per posizione In questo caso, gli argomenti devono essere nello stesso ordine indicato dalla lista delle keyword.

Per keyword In questo caso l'ordine non ha importanza e si fa precedere ogni argomento dalla keyword di competenza.

Per esempio, se l'istruzione MIOCOMANDO legge da un file e scrive verso un altro, il template degli argomenti viene a essere:

```
FROM,TO
```

Si può usare il comando specificando gli argomenti per posizione:

```
MIOCOMANDO file-di-input file-di-output
```

oppure usando le keyword:

```
MIOCOMANDO FROM file-di-input TO file-di-output
MIOCOMANDO TO file-di-output FROM file-di-input
```

Si possono anche combinare le indicazioni degli argomenti per keyword e per posizione, come nel caso seguente

```
MIOCOMANDO file-di-input TO file-di-output
```

dove l'argomento FROM viene fornito per posizione, e l'argomento TO per keyword. La forma seguente è invece scorretta:

```
MIOCOMANDO file-di-output FROM file-di-input
```

poiché il comando presume che il file di output sia il primo argomento posizionale (cioè il file FROM).

Se l'argomento non è una parola singola (cioè, se contiene o è "delimitata" da spazi), deve essere racchiuso tra virgolette (''). Se l'argomento ha lo stesso nome di una delle keyword, dev'essere racchiuso anch'esso tra virgolette. Per esempio, il seguente

```
MIOCOMANDO ''nome file'' TO ''from''
```

fornisce il testo "nome file" come argomento FROM, e il nome del file "from" come argomento TO.

Le keyword nelle liste di argomenti hanno associati certi qualificatori, che sono rappresentati da una barra (/) e da una lettera specifica. Riportiamo qui di seguito il significato dei vari qualificatori:

/A L'argomento è necessario e non può essere omesso.
/K L'argomento deve essere fornito con la keyword e non può essere usato in senso posizionale.
/S La keyword funziona come un pulsante di attivazione e disattivazione (un toggle) e non ha argomenti.

I qualificatori A e K possono essere combinati, e perciò il template

```
DRIVE/A/K
```

significa che si deve fornire sia l'argomento sia la keyword DRIVE.

In alcuni casi non deve essere fornita nessuna keyword. Per esempio, il comando DELETE riceve dall'AmigaDOS l'ordine di cancellare un certo numero di file. In questo caso, si omette semplicemente il valore della keyword, ma rimangono nel template le virgole normalmente usate per separare le keyword. Così, il template per il comando DELETE, che può ricevere un massimo di dieci file, è:

```
,,,,,,,,,
```

Infine, si consideri il comando TYPE. Il suo template è

```
FROM/A,TO,OPT/K
```

e significa che il primo argomento può essere specificato per posizione o per keyword e che è necessario. Il secondo argomento (TO) è opzionale e la sua keyword può essere omessa. L'argomento OPT è opzionale, ma se viene indicato, deve essere fornito con la keyword. Quindi, tutte le forme seguenti del comando TYPE sono valide:

```
TYPE nome_del_file
TYPE FROM nome_del_file
TYPE nome_del_file TO file_di_output
TYPE nome_del_file file_di_output
TYPE TO file_di_output FROM nome_del_file OPT n
TYPE nome_del_file OPT n
TYPE nome_del_file OPT n TO file_di_output
```

Nonostante questo manuale fornisca una lista di tutti gli argomenti dei vari comandi, è possibile visualizzare il template degli argomenti semplicemente digitando il nome del comando seguito da un punto di domanda (?).

Se gli argomenti forniti non combaciano con quelli del template, la maggior parte dei programmi visualizza semplicemente il messaggio "Bad args" o "Bad arguments" (argomenti non corretti) e si ferma; di conseguenza bisogna riscrivere il comando e gli argomenti. Per visualizzare un aiuto sullo schermo riguardo agli argomenti che un comando si aspetta di ricevere, si può aggiungere sempre un punto di domanda (?).

1.5 Processo di convalida

Quando un disco viene introdotto in un drive per la prima volta, l'AmigaDOS crea un processo a bassa priorità che convalida l'intera struttura del disco. Finché il processo non è terminato non si possono creare file sul disco, mentre è possibile leggerli.

Quando il processo di convalida è stato completato, l'AmigaDOS controlla se è stata impostata la data. Per impostare la data e l'ora, si deve usare il comando DATE. Se non è stata specificata una data di sistema, l'AmigaDOS imposta la data e il tempo di sistema uguale a quelli del file più recente presente sul disco appena inserito. Questo garantisce che le nuove versioni dei file abbiano sempre date più recenti, nonostante il fatto che l'ora e la data siano in effetti sbagliate.

Se si richiede l'ora e la data prima che la convalida sia terminata, l'AmigaDOS le mostra come non impostate. Si può quindi aspettare la fine del processo di convalida, oppure usare il comando DATE per introdurre il tempo e la data corretti. La convalida dovrebbe verificarsi una volta sola; in ogni caso, non dura mai più di un minuto.

1.6 Comandi d'uso più comune

Questo manuale descrive l'AmigaDOS e i suoi comandi. L'Interfaccia Linea Comando (CLI, acronimo di Command Line Interface) legge i comandi digitati in una finestra CLI e li trasforma in azioni eseguite dal computer. In questo senso, il CLI è simile alle interfacce utente più tradizionali: si inseriscono dei comandi e il sistema visualizza in risposta alcune linee di testo.

Dal momento che alla maggior parte degli utenti è sufficiente il Workbench, che tra l'altro è particolarmente "user-friendly", il suo disco viene fornito con il CLI "disabilitato". Per utilizzare i comandi descritti in questo manuale bisogna prima "abilitare" il CLI. Quindi è necessario disporre, nel proprio Workbench, di una nuova icona, chiamata "CLI". Quando viene selezionata e aperta quest'icona, si rende disponibile una finestra CLI, che può essere utilizzata per indirizzare i comandi in forma testuale direttamente all'AmigaDOS.

Come abilitare l'Interfaccia Linea Comando (CLI)

Attivate il vostro computer inserendo i dischetti del Kickstart (se si tratta di un Amiga 1000) e del Workbench. Aprite l'icona del disco e in seguito quella di "Preferences". Nella parte sinistra dello schermo, a circa due terzi di altezza partendo dal basso, noterete la scritta "CLI" con un riquadro per "ON" e uno per "OFF". Selezionate "ON" e in seguito puntate su "Save" (in basso a destra) per abbandonare Preferences.

Come aprire una finestra CLI

Per usare i comandi CLI, occorre aprire una finestra CLI. Aprendo il cassetto "System" dovrebbe essere visibile l'icona del CLI (un rettangolo che contiene un "1>"). Apritela.

Uso del CLI

Per usare l'interfaccia utente CLI selezionatene la finestra e digitate i comandi CLI desiderati. La finestra CLI, o le finestre, possono essere dimensionate o spostate esattamente come le altre finestre. Per chiudere la finestra, digitate "ENDCLI".

Workbench e CLI, relazioni e differenze

Scrivete "DIR" per visualizzare la lista dei file e delle directory presenti nella directory corrente. Si tratta della lista dei file che costituiscono il vostro Workbench. Si può notare che in questa directory sono presenti più file di quante icone siano visualizzate nel Workbench. Il Workbench visualizza solo i file "X" che dispongono anche di un corrispondente file "X.info". I file ".info" contengono tutte le informazioni necessarie per gestire le icone.

Per esempio, il programma diskcopy è individuato da due file: il file "DiskCopy" contiene il programma vero e proprio, mentre il file "DiskCopy.info" contiene le informazioni per il Workbench che lo riguardano. Nel caso di file con i dati di una figura grafica come "mount.pic", il file "mount.pic.info" contiene le informazioni che riguardano l'icona e il nome del programma (di default) che dovrebbe elaborarlo (GraphiCraft). In questo caso, quando l'utente "apre" il file di dati (mount.pic) il Workbench esegue il programma (GraphiCraft) e gli comunica il nome del file di dati (mount.pic).

Le subdirectory dell'AmigaDOS corrispondono ai cassetti del Workbench.

I device con accesso diretto ai blocchi come i dischi (df0:) corrispondono alle icone dei dischi che avete già visto.

Non tutti i programmi o i comandi possono essere eseguiti sia dall'ambiente Workbench sia da quello CLI. Nessuno dei comandi CLI descritti nel capitolo 2 di questo manuale può essere eseguito da Workbench. Per esempio, nella versione 1.1 esistono due diversi comandi DISKCOPY. Quello nella directory :c/ viene eseguito da AmigaDOS (CLI), mentre quello nella directory (cassetto) system viene eseguito da Workbench.

Introduzione ad alcuni comandi dell'AmigaDOS

Benché tutti i comandi disponibili attraverso il CLI siano spiegati in dettaglio nella parte di consultazione rapida di questo manuale, la maggior parte di noi fa scarso uso delle opzioni avanzate disponibili. Si è quindi provveduto a fornire un sommario che mostri i vari comandi nella forma più usata.

I comandi elencati qui di seguito (accompagnati dai nomi reali dei comandi dell'AmigaDOS) permettono di svolgere operazioni come:

- Copiare un disco (DISKCOPY)
- Formattare un nuovo disco (FORMAT)
- Installare un disco; creare un disco CLI (INSTALL)
- Cambiare nome a un disco (RELABEL)
- Ispezionare la directory di un disco (DIR)
- Ottenere informazioni riguardo ai file (LIST)
- Prevenire cancellazioni accidentali (PROTECT)
- Ottenere informazioni riguardo al file system (INFO)
- Cambiare la directory corrente (CD)
- Impostare l'ora e la data (DATE)
- Ridirigere l'output di un comando (>)
- Visualizzare un file di testo sullo schermo (TYPE)
- Cambiare il nome di un file (RENAME)
- Cancellare un file (DELETE)
- Creare una nuova directory (MAKEDIR)
- Copiare i file in un sistema dotato di due disk drive (COPY)
- Copiare i file in un sistema dotato di un solo disk drive (COPY)
- Ricercare i file in un disco (DIR OPT A)
- Eseguire qualcosa automaticamente alla partenza del sistema (usando la Startup-Sequence)
- Comunicare all'AmigaDOS dove orientare la ricerca (ASSIGN)
- Aprire una nuova finestra CLI (NEWCLI)
- Chiudere una finestra CLI esistente (ENDCLI)

Tutti i comandi elencati nella pagina precedente, richiedono che abbiate attivato il sistema con un disco CLI invece che con un disco Workbench, oppure che abbiate attivato il CLI per mezzo del programma Preferences e che siate entrati nel CLI usando questa via.

La sequenza per attivare il CLI è stata fornita precedentemente in questo manuale.

Per i nuovi utenti

A coloro che si avvicinano all'Amiga per la prima volta consigliamo di leggere e provare ognuno di questi comandi, uno alla volta. Ogni comando mostrato qui di seguito, lascia il vostro disco-prova in condizione di ricevere il comando immediatamente seguente, che agirà esattamente come viene indicato. Più tardi, quando avrete acquisito confidenza col sistema, i titoli dei paragrafi serviranno per rinfrescarvi la memoria.

Come iniziare

Prima di iniziare questa sezione, siate sicuri di possedere due dischi a doppia faccia vuoti, e di disporre del disco con il Workbench o il CLI. Proteggete in scrittura il disco originale e abilitate in scrittura i due dischi vuoti. La maggior parte dei comandi illustrati sono dati nella forma utilizzata da chi possiede un solo disk drive; talvolta comunque, viene fornita anche la versione per due drive.

I comandi illustrati in questa sezione sono rientrati rispetto al margine sinistro e sono scritti in maiuscolo solo per distinguerli dal resto del testo. L'AmigaDOS riconosce sia i comandi in maiuscolo sia quelli in minuscolo. Dopo aver digitato ogni comando, occorre premere il tasto RETURN per ridare il controllo all'AmigaDOS.

Nella sezione che segue, le notazioni "df0:" e "drive 0" si riferiscono al drive interno dell'Amiga. La notazione "df1:" si riferisce al primo disk drive aggiuntivo da 3,5 pollici.

Occasionalmente sulla linea del comando da inserire può trovarsi un punto e virgola (;). Il testo che segue il punto e virgola viene trattato dall'AmigaDOS come un commento. Dal momento che l'AmigaDOS ignora il resto della linea, non c'è bisogno che scriviate assieme al comando il commento, che è stato inserito solo per vostra informazione.

È possibile ottenere per la maggior parte dei comandi una limitata forma di aiuto scrivendo il nome del comando stesso seguito da un punto di

domanda (?) e premendo RETURN. Viene mostrato il "template" del comando, contenente la sequenza dei parametri attesi e le keyword riconosciute.

Copiare un disco

Potete usare questa sequenza per creare una copia di sicurezza (backup) del disco sistema originale o di qualunque altro disco.

Per un sistema con 1 drive

```
DISKCOPY FROM df0: TO df0:
```

Per un sistema con 2 drive

```
DISKCOPY FROM df0: to df1:
```

Seguite le istruzioni così come appaiono. Per un sistema con un solo drive vi viene chiesto di inserire il disco originale (FROM). Poi, con il progredire della copia, l'AmigaDOS chiede l'inserimento del disco copia (TO), e così via, scambiando di continuo il disco originale con quello copia fino a che non termina la duplicazione. Per un sistema con due drive vi viene chiesto di inserire il disco originale in df0: (il drive interno) e il disco sul quale deve essere effettuata la copia in df1: (il primo drive aggiuntivo).

Togliete il disco originale e mettetelo in un posto sicuro. Lasciate la copia abilitata in scrittura in modo che possiate memorizzarvi delle informazioni. Inserite la copia appena effettuata nel drive interno e fate ripartire il sistema (vedere il *Manuale di introduzione all'Amiga* fornito con la macchina, per il processo di inizializzazione del sistema).

Dopo la nuova inizializzazione, entrate nuovamente in modo CLI. Se ripartite con un disco CLI vi troverete automaticamente nell'ambiente CLI. Se state usando un disco Workbench dovrete aprire l'icona CLI presente nel cassetto system.

Formattare un disco

Per provare il comando di formattazione dovete avere il disco sistema nel drive 0 e disporre di un disco vuoto.

Alcune volte invece di copiare semplicemente un disco potreste voler preparare un disco di dati per il vostro sistema. Più tardi potreste copiare alcuni particolari file su questo disco di dati. Formattate quindi il vostro disco

vuoto usando il comando FORMAT

```
FORMAT DRIVE df0: NAME ''QualunqueNome''
```

Seguite le istruzioni segnalate dal sistema. Potete formattare i dischi sia nel drive 0 (df0:, interno all'Amiga) che nel drive aggiuntivo.

Dopo che la formattazione è stata completata, aspettate che la luce del drive, che ne indica l'attività, si spenga e togliete il disco appena formattato. Inserite nuovamente il disco sistema. Il disco formattato può essere usato per contenere file di dati ma non può essere usato per attivare il sistema.

Installare un disco

Per provare questo comando, dovete avere il vostro disco sistema nel drive 0 e un disco appena formattato. Sono possibili diversi modi per creare un disco CLI; ne illustreremo due.

Un disco installato è un disco che può essere usato per far partire il sistema (per l'Amiga 1000 l'inserimento del disco installato può avvenire solo dopo l'avvenuto caricamento del Kickstart). Potete cambiare un disco formattato in un disco CLI digitando il comando:

```
INSTALL ?
```

Nota: *per usare questo comando in un sistema con un solo drive, DOVETE usare il punto di domanda! Altrimenti l'AmigaDOS tenterà di eseguire il comando sul disco attualmente nel drive 0.*

L'AmigaDOS risponderà con:

```
DRIVE/A
```

Togliete il disco sistema e inserite il disco formattato. Poi digitate:

```
df0:
```

e premete RETURN. L'AmigaDOS copierà i settori appropriati sul disco. Ora, se aspettate che la luce del drive si spenga ed eseguite un reset completo della macchina (CTRL-Amiga-Amiga) vi troverete, quando il sistema ha terminato la sua inizializzazione, direttamente nel CLI invece che nel Workbench.

Il vostro disco formattato contiene ora il CLI e niente altro. Ciò significa che, nonostante l'interprete sia presente, nessuno dei comandi illustrati in questa sezione può essere eseguito. Un CLI ha bisogno di un certo numero di file perché i comandi possano essere eseguiti. Tutti i file dei comandi sono posti nella directory C del disco originale.

Il secondo modo di produrre un disco CLI è più utile visto che lascia la directory dei comandi CLI intatta. Di seguito è riportato il processo

passo-per-passo per cambiare una copia modificabile di un disco Workbench in un disco CLI:

1. Copiate il vostro disco con il Workbench.
2. Aprite il CLI come descritto più volte.
3. Premete il pulsante di selezione del mouse nella finestra CLI e digitate il comando:

```
RENAME FROM s/startup-sequence TO s/no-startup-sequence
```

Ora, se aspettate che la luce del drive si spenga ed eseguite un reset completo della macchina potrete constatare come la vostra copia del Workbench sia diventata un disco CLI. Per ripristinare il Workbench, eseguite il comando RENAME nuovamente ma con i due nomi di file cambiati di posto. Se l'AmigaDOS non riesce a trovare il file con l'esatto nome di "startup-sequence" nella directory "s", entra in CLI e aspetta che venga digitato un comando.

Cambiare nome a un disco

Prima di provare questo comando ponete il disco sistema nel drive 0. Se, dopo aver copiato o formattato un disco, non siete soddisfatti del nome che gli avete dato, lo potete cambiare usando il comando RELABEL:

```
RELABEL QualunqueNome: AltroNome
```

In questo esempio ci si è riferiti al disco che è stato formattato con il suo nome di volume. Vi verrà chiesto di inserire il volume "QualunqueNome" in un drive qualsiasi in modo che il comando RELABEL gli possa cambiare il nome.

Dopo l'esecuzione del comando togliete il disco e inserite nuovamente il disco sistema. Il disco che avete rimosso possiede ora il nuovo nome.

Lettura della directory

Prima di provare questo comando inserite il disco sistema nel drive 0. Potete ispezionare il contenuto di una directory per mezzo del comando

`DIR` oppure `DIR df0:`

In questa forma viene visualizzato il contenuto della directory corrente. È possibile accedere al contenuto di una directory differente indicandone il

percorso d'accesso (pathname). Per esempio il comando:

`DIR df0:c` oppure `DIR c`

visualizza il contenuto di c (dir) dal drive 0. Le directory sono equivalenti ai cassetti che potete vedere sullo schermo del Workbench, quando è visibile.

Potete vedere le directory di un disk drive aggiuntivo, se lo possedete, specificandone il nome. Per esempio:

```
DIR df1:
```

visualizza il contenuto del disco inserito nel drive 1.

È anche possibile vedere le directory di un disco che non è inserito in nessun drive, specificandone il nome di volume. Per esempio, il contenuto del disco appena formattato e il cui nome è stato cambiato può venire visualizzato per mezzo del comando:

```
DIR AltroNome:
```

Uso del comando LIST

Inserite il disco sistema nel drive 0 per poter provare questo comando. Il comando DIR fornisce i nomi dei file che si trovano nella vostra directory. Il comando LIST fornisce ulteriori informazioni riguardo a quei file. Digitate il comando:

`LIST` oppure `LIST df0:`

L'AmigaDOS fornisce informazioni riguardo a tutti i file presenti nella directory corrente, tra cui la dimensione del file, se il file può essere o meno cancellato, se è un file o una directory, e quali sono il giorno e l'ora della sua creazione.

Se insieme a LIST si specifica il nome di una directory, il comando riporterà le informazioni sui file contenuti in quella directory:

```
LIST c
```

I "rwed" sono chiamati flag di protezione e stanno per Read (leggere), Write (scrivere), Execute (eseguire) e Delete (cancellare). Se ogni flag è impostato, usando il comando PROTECT, si suppone che il file sia leggibile, alterabile, eseguibile e cancellabile. Nella versione attualmente in uso, l'AmigaDOS presta attenzione solo al flag di cancellazione (d). Se la "d" non

viene mostrata nella colonna dei flag di protezione di un file, l'AmigaDOS non lo cancellerà durante l'esecuzione del comando DELETE.

Uso del comando PROTECT

Per provare questo comando il disco sistema deve essere inserito nel drive 0. Questo comando protegge un file da cancellazioni accidentali, oppure toglie la protezione. Provate i seguenti comandi:

```
DATE > mio.file
PROTECT mio.file
LIST mio.file
```

Potete vedere, usando il comando LIST, come i flag di protezione ora siano posti a "- - - -". Se ora cercate di eseguire

```
DELETE file.mio
```

l'AmigaDOS risponde con:

```
''Not Deleted - file is protected from deletion''
```
(Non Cancellato - il file è protetto dalla cancellazione)

Per abilitare nuovamente la cancellabilità del file:

`PROTECT mio.file d` oppure `PROTECT mio.file rwed`

Le informazioni sul file system

Il vostro disco sistema deve trovarsi nel drive 0. Digitate il comando:

```
INFO
```

Questa istruzione vi comunicherà quanto spazio è occupato e quanto è libero nei vostri dischi, se sono read-only (protetti in scrittura) o read-write (non protetti), e i nomi dei volumi. Potete creare più spazio libero in un disco cancellando alcuni file e potete cambiarne il nome di volume usando il comando RELABEL. Se desiderate informazioni su un disco che al momento non si trova nel vostro unico drive, digitate il comando:

```
INFO ?
```

l'AmigaDOS risponde con:

```
none:
```

L'AmigaDOS ha caricato il comando INFO dal vostro disco CLI e ne ha mostrato il template. La risposta "none:" (nessuno) vi dice che dovete premere solo il tasto RETURN per eseguire il comando. Togliete il vostro disco CLI e inserite il disco sul quale desiderate operi il comando INFO. Aspettate che la luce del drive si accenda e poi si spenga, quindi premete RETURN. l'AmigaDOS vi fornirà ora le informazioni riguardanti il disco appena inserito. Questo trucchetto funziona sia per il comando DIR sia per INFO.

Cambiare la directory corrente

Fino a questo punto siamo rimasti alla "radice" o, sarebbe meglio dire, al livello gerarchico più elevato delle directory del disco. Potete trovare maggiori informazioni sulla struttura ad albero delle directory nella sezione 1.3 del manuale. Per vedere a quale livello dell'albero delle directory vi trovate attualmente, usate il comando:

```
CD
```

Per cambiare la directory corrente, dovete comunicare al sistema quale deve diventare la nuova. Per esempio, quando avete eseguito il comando DIR per df0:, il disco CLI ha rivelato l'esistenza di una c (dir). Se volete che questa directory diventi quella corrente, digitate il comando:

`CD c` oppure `CD df0:c`

Ora, se richiedete l'esecuzione del comando DIR, vi verrà mostrato il contenuto di questo livello del filing system. Il comando CD (senza parametri) visualizza il nome della directory corrente. Potete tornare alla directory radice (il livello più alto) digitando:

```
CD :
```

per il volume corrente (se vi riferite ai dischi per mezzo del nome di volume) oppure

```
CD df0:
```

per il disco presente nel drive interno.

Impostare la data e l'ora.

Potete impostare l'orologio dell'AmigaDOS usando il comando DATE:

```
DATE 12:00:00 12-Oct-88
```

L'orologio di sistema misurerà il trascorrere del tempo a partire da quest'ora e da questa data.

Ridirigere l'output di un comando

Il disco sistema dev'essere nel drive 0 per provare questo comando. Normalmente l'output di tutti i comandi viene inviato allo schermo del monitor. Potete cambiare la destinazione verso cui il sistema dirige l'output usando il comando di ridirezione ">". Il segno di maggiore significa che l'output viene mandato al file specificato dopo il segno. Ecco un esempio:

```
DATE > file.data
```

Eseguite il comando in modo da poter usare successivamente il file.data. Questo comando crea (o sovrascrive) un file chiamato "file.data" nella directory corrente. Oppure, giusto per avere qualcosa nel disco formattato chiamato "AltroNome", digitate quanto segue:

```
DATE > AltroNome:file.data
```

L'AmigaDOS vi richiede di inserire il volume con quel nome. Dopo che la luce del drive si è spenta, togliete AltroNome e inserite di nuovo il disco sistema. Ora scrivete il comando:

```
DIR AltroNome:
```

Vi verrà richiesto un'altra volta di inserire AltroNome in un drive qualunque. L'AmigaDOS visualizza quindi la directory di questo disco che ora contiene un file chiamato file.data.
Ora rimettete il disco sistema o il Workbench nel drive 0.

Visualizzare un file di testo sullo schermo

Potete vedere il contenuto di un file di testo usando il comando TYPE:

```
TYPE file.data
```

Questo comando visualizza qualunque cosa sia presente nel file. Se desiderate fermare l'output momentaneamente per leggere qualcosa sullo

schermo, premete la barra spaziatrice. Per riprendere la visualizzazione premete il tasto BACKSPACE. Se desiderate terminare l'esecuzione del comando TYPE, mantenete premuto il tasto CTRL mentre premete anche il tasto C.

Se volete verificare il contenuto di file.data presente nell'altro disco, potete eseguire il comando:

```
TYPE AltroNome:file.data
```

Cambiare il nome di un file

Il disco sistema deve trovarsi nel drive 0. Potete cambiare il nome di un file usando il comando RENAME:

```
RENAME FROM file.data TO file.nuovo
```

oppure

```
RENAME file.data file.nuovo
```

Ora usate il comando TYPE per verificare che il nuovo nome si riferisca allo stesso contenuto.

```
TYPE file.nuovo
```

Notate che la forma alternativa del comando non richiede l'uso delle keyword FROM e TO. La maggior parte dei comandi dell'AmigaDOS hanno una forma alternativa, abbreviata rispetto a quella mostrata in questa sezione. La forma più lunga è stata usata principalmente per illustrare cosa fa effettivamente il comando. Riferirsi al sommario per l'elenco delle forme di comando alternative che sono disponibili.

Cancellare i file

Il disco sistema si deve trovare nel drive 0. Può darsi che vi troviate a lavorare con diverse versioni di un programma o di un file di testo, e desideriate cancellare le versioni di cui non avete più bisogno. Il comando DELETE vi permette di cancellare i file e di liberare lo spazio che utilizzavano per usarlo in seguito.

Nota: *se cancellate un file non c'è modo di recuperarlo. Assicuratevi di volerlo veramente cancellare.*

Qui di seguito è riportata una sequenza di comandi che crea un file usando

il comando di ridirezione, lo visualizza per verificare che esiste realmente e poi lo cancella.

```
DIR > file.dir
TYPE file.dir
DELETE file.dir
TYPE file.dir
```

L'AmigaDOS risponde al comando finale della sequenza con il messaggio:

`Can't Open file.dir` (il file non può essere aperto)

per indicare che il file non può essere trovato dal momento che è stato cancellato.

Copiare i file

Il disco sistema, contenente il Workbench o il CLI, dev'essere nel drive 0. In un sistema con due drive la copia dei file è semplice:

```
COPY FROM df0:sorgente TO df1:destinazione
```

oppure

```
COPY df0:sorgente df1:destinazione
```

In un sistema con un solo drive, la copia dei file è leggermente più complessa. Dovete copiare alcuni file di sistema dal vostro disco nella memoria principale, che è meglio conosciuta come device RAM: oppure "ramdisk". Quindi copiare i(1) file nel ramdisk, porre come directory corrente il ramdisk e poi copiare dal ramdisk verso il disco di destinazione. Di seguito viene riportata la sequenza tipica:

```
COPY df0:c/cd RAM:
COPY df0:c/copy RAM:
CD RAM:
```

Inserite il disco contenente il file da copiare nel drive (in questo esempio ci serviremo del disco sistema che si trova già nel drive). Digitate:

```
COPY df0:c/execute ram:execute
```

oppure

```
COPY df0:c/execute execute
```

oppure

```
COPY df0:c/execute ram:
```

Estraete il disco sorgente e inserite quello destinazione; quindi digitate:

```
COPY ram:execute df0:execute
```

oppure

```
COPY execute df0:execute
```

Estraete il disco destinazione e inserite nuovamente il disco sistema, poi scrivete:

```
CD df0:
```

e vi ritrovate nello stesso punto dal quale siete partiti. L'unico altro comando che potreste voler eseguire è:

```
DELETE RAM:cd RAM:copy RAM:execute
```

che restituisce la memoria del ramdisk al sistema per altri impieghi.

Creare una nuova directory

Potete creare una nuova directory all'interno della directory corrente usando il comando MAKEDIR:

```
MAKEDIR nuovo.cassetto
```

Se ora digitate il comando DIR, tra gli altri nomi apparirà:

```
nuovo.cassetto (dir)
```

Potete anche usare il comando RENAME per muovere un file da una directory a un'altra dello stesso disco. Il comando:

```
MAKEDIR nuovo.cassetto
RENAME FROM file.nuovo TO nuovo.cassetto/file.nuovo
```

muove il file dalla directory corrente a quella appena creata. Per verificare se il file è stato effettivamente spostato, impartite il comando:

```
DIR
```

Poi proseguite con:

```
DIR nuovo.cassetto
```

L'AmigaDOS cerca nella directory nuovo.cassetto, e mostra che il file chiamato "file.nuovo" si trova effettivamente lì.

C'è il mio file su questo disco?

Il disco sistema si deve trovare nel drive 0. Può accadere che desideriate vedere tutto il contenuto di un disco, invece di una sola directory alla volta. Potete usare il comando DIR con una delle sue opzioni:

```
DIR OPT A
```

che visualizza tutte le directory e subdirectory presenti nel disco. Ricordate l'uso della combinazione <SPACE BAR><BACKSPACE> per fermare e riprendere la visualizzazione. L'effetto del tasto BACKSPACE è uguale a quello di RETURN, ma con RETURN lo scrolling viene interrotto da una linea vuota.

Per dare uno sguardo più accurato al contenuto di un disco, potreste ridirigere l'output verso un file:

```
DIR > dir.mio.disco OPT A
```

Notate che il comando per la ridirezione dell'output e il nome del file DEVONO venire prima dell'opzione riferita al comando DIR.

Ora, se lo desiderate, potete vedere il contenuto del file "dir.mio.disco" facendo uso del comando TYPE, della barra spaziatrice per fermare momentaneamente la visualizzazione e del tasto BACKSPACE per riprenderla. Oppure potete usare ED per vedere il file nel modo seguente:

```
ED dir.mio.disco
```

- Usate i tasti cursore per scorrere il file avanti e indietro.
- Usate la combinazione di tasti ESC e T <RETURN> per muovervi all'inizio del file. Questa combinazione può essere definita come "ESC-T", il che significa ESC seguito da T.
- Usate la combinazione di tasti ESC-B <RETURN> per muovervi alla fine del file.
- Usate la combinazione ESC-M seguito da un numero <RETURN> per muovervi a un numero di linea specifico.
- Usate ESC-Q <RETURN> per uscire dall'ED senza cambiare il file oppure

- Usate ESC-X <RETURN> per eseguire qualunque modifica nel file originale.

Il capitolo 3 di questo manuale fornisce informazioni dettagliate per l'uso di ED.

Operazioni automatiche in fase di attivazione del sistema

Nella directory "s" del disco sistema è presente un file chiamato Startup-Sequence (sequenza di partenza) che è uno di quelli usati dal comando EXECUTE (per via di questa particolarità vengono chiamati "execute file"). Contiene una sequenza di comandi CLI che l'AmigaDOS esegue ogni volta che il sistema viene inizializzato. Gli ultimi due comandi nel file Startup-Sequence del vostro disco del Workbench sono LoadWB (carica il programma Workbench) e ENDCLI il quale cede il controllo al Workbench. Potete costruire il vostro file Startup-Sequence usando i programmi ED o EDIT. Il compendio del comando EXECUTE e la sezione "didattica" nel capitolo 2 forniscono i particolari che riguardano i vari comandi inseribili in questi file. Notate che il file Startup-Sequence può essere usato per mandare un programma in autoesecuzione.

ATTENZIONE: se decidete di modificare il file Startup-Sequence, alterate solo la copia del disco sistema, e non l'originale.

Assegnare i percorsi di ricerca dell'AmigaDOS

Il disco sistema, come sempre, deve trovarsi nel drive 0 per poter provare il comando che stiamo per descrivere. Potrebbe capitare che desideriate usare un disco differente da quello di sistema e continuare il vostro lavoro. Per esempio, può succedere che inizializziate il sistema con un disco del Workbench e poi che vogliate usare un disco CLI. Se il disco CLI possiede una directory (per esempio c (dir)) contenente i comandi eseguibili che desiderate utilizzare, potete cambiare il disco usando il comando ASSIGN.

Se non usate ASSIGN, per far eseguire i comandi dovrete scambiare i dischi. Di seguito è riportato un esempio che non fa uso del comando ASSIGN. L'intento è di scambiare i dischi e di usare il disco "mio.disco:" come disco sistema. Qualunque file non necessario deve già essere stato cancellato in modo da fornire ulteriore spazio di lavoro. Inserite:

```
CD mio.disco:
```

L'AmigaDOS risponde con il messaggio "insert mio.disco in any drive", che significa "inserisci mio.disco in un drive qualunque". Eseguite quanto richiesto, quindi digitate:

```
DIR
```

L'AmigaDOS richiede "insert Workbench (o qualunque altro nome abbia il vostro disco del Workbench) in any drive". Infatti sa, dal momento dell'inizializzazione, che il comando DIR si trova nella directory c del disco del Workbench. L'AmigaDOS legge il comando DIR e poi chiede nuovamente "insert mio.disco in any drive". Qualunque altro comando dell'AmigaDOS ha come risultato uno scambio di dischi. Per risolvere il problema usate il comando ASSIGN come segue:

```
ASSIGN c: mio.disco:c
```

L'AmigaDOS richiede l'inserimento del disco "mio.disco" e d'ora in poi tutti i comandi vengono cercati nella directory c di questo disco. L'AmigaDOS quindi non richiederà più l'introduzione del Workbench per eseguire i comandi.

Una volta impartito il comando, probabilmente vorrete digitare:

```
CD mio.disco
```

Oltre alla directory c, ci sono altre voci che possono venir assegnate dall'AmigaDOS al nostro disco. Se digitate il comando

```
ASSIGN LIST
```

potrete vedere la lista di queste voci. Se fate eseguire un programma che utilizza la porta seriale (serial device) o quella parallela (parallel device), l'AmigaDOS cercherà nella directory attualmente assegnata a DEVS: per individuare i relativi device. Se tutte le directory di sistema si trovano nel disco del Workbench, potete evitare che l'AmigaDOS vi chieda in continuazione di inserirlo, eseguendo un file di comandi che assegni tutto alle directory del nuovo disco. Il contenuto di questo "execute file" per il disco "mio.disco" potrebbe essere:

```
ASSIGN SYS: mio.disco:
ASSIGN S: mio.disco:s
ASSIGN DEVS: mio.disco:devs
ASSIGN L: mio.disco:l
ASSIGN FONTS: mio.disco:fonts
ASSIGN LIBS: mio.disco:libs
```

Per creare l'execute file, usate il comando:

```
COPY FROM * TO riassegna
```

Poi digitate le linee precedenti, che contengono i comandi di ASSIGN. Dopo aver scritto l'ultima linea, usate la combinazione di tasti CTRL-\ per terminare l'immissione del file. Il carattere "*" indica la tastiera e la finestra CLI corrente. Questo metodo è una via alternativa rispetto all'impiego di ED oppure di EDIT.

Creare un nuovo CLI

L'AmigaDOS è un sistema operativo multitasking, cioè è in grado di eseguire contemporaneamente molti processi. Potete avere più finestre aperte nello stesso momento, ognuna con la propria directory corrente e ognuna che esegue un comando diverso. Potete creare un nuovo CLI per mezzo del comando NEWCLI:

```
NEWCLI
```

Questo comando apre una finestra separata, con un prompt che identifica il processo corrente. Per esempio, se la prima finestra ha come prompt:

```
1>
```

allora il nuovo CLI avrà:

```
2>
```

Potete spostare la nuova finestra, allargarla, restringerla e così via come per qualunque altra finestra. Per eseguire un comando nel nuovo CLI, premete il tasto sinistro del mouse quando è situato all'interno della finestra. Qualunque cosa scriviate, viene mandata all'ultima finestra nella quale avete premuto il pulsante di selezione del mouse. Provate quanto segue:

1. Selezionate la finestra 1, poi scrivete:

```
DIR df0:
```

2. Selezionate velocemente la finestra 2 e scrivete:

```
INFO
```

Entrambi i CLI lavorano nello stesso momento per soddisfare le vostre richieste, dimostrando le capacità multitasking dell'Amiga. Noterete che

non siete limitati a due CLI, ma, se lo desiderate e disponete di memoria sufficiente, potete aprirne anche 20.

Chiudere un CLI

Potete terminare un processo CLI e chiudere la relativa finestra per mezzo del comando ENDCLI. Selezionate la finestra CLI che desiderate chiudere e digitate:

```
ENDCLI
```

Questo è tutto.

Conclusioni

La serie di comandi appena esposta, rappresenta una panoramica sulle possibilità offerte dai comandi dell'AmigaDOS impartiti da CLI. Vi sono però diversi comandi che non sono stati presi in considerazione nella trattazione precedente. Inoltre, la maggior parte dei comandi descritti dispongono di altri "template" (i modi in cui potete inserire un comando) e opzioni che non sono state illustrate.

Il capitolo 2 di questo manuale contiene una sezione di consultazione che illustra il template per ognuno dei comandi dell'AmigaDOS; potete quindi leggere la descrizione di ogni comando per ottenere maggiori informazioni. Una volta che vi siete impadroniti dei comandi e dei diversi modi in cui è possibile usarli, vi può tornare utile, per ricordare i comandi disponibili, l'indice di consultazione rapida presente alla fine del capitolo.

1.7 Convenzioni adottate

Nel capitolo 2 del manuale, nella descrizione del "formato" dei comandi dell'AmigaDOS, viene adottata la seguente notazione:

<nome> Indica il nome del parametro che deve accompagnare il comando. Esempio: EXECUTE <filecomando> dove il nome del file contenente i comandi è un parametro obbligatorio.

[ ] Le parentesi quadre indicano un parametro opzionale. Il comando non ha bisogno che tale parametro venga indicato,

ma se viene fornito aggiunge ulteriori informazioni utili all'AmigaDOS per sapere come dev'essere eseguito il comando stesso.

| Una barra verticale indica che potete scegliere una o l'altra delle alternative indicate. Esempio: DIR [OPT A|I|AI]. L'esempio indica che si può specificare A, I oppure AI.

<nome>* Indica una o più ricorrenze del nome di un parametro; se indicate più di un parametro di questo tipo, ognuno deve essere separato dal successivo con almeno uno spazio bianco.

I comandi dell'AmigaDOS non usano nessun segno di interpunzione, come virgole o punti, ma utilizzano gli spazi bianchi per separare tra loro i diversi parametri. Non confondete le informazioni che riguardano il "formato" di un comando con il suo "template". La sezione 1.4.5 spiega il template dei comandi.

Capitolo 2

I comandi dell'AmigaDOS

Questo capitolo è suddiviso in due parti: la prima descrive i comandi per l'utente disponibili sull'Amiga, la seconda quelli per i programmatori. I comandi per l'utente appartengono a diverse categorie: trattamento dei file, controllo del CLI, controllo delle sequenze di comandi, gestione del sistema e della memoria. La parte 1 raccoglie le descrizioni dei comandi in ordine alfabetico, e viene indicato per ognuno il formato, il template, lo scopo, la spiegazione e un esempio applicativo. La parte 2 mantiene la stessa organizzazione.

Il capitolo inizia illustrando una serie di termini poco familiari. Alla fine del capitolo si trova una lista per la consultazione rapida dei vari comandi e funzioni.

2.1 I comandi dell'AmigaDOS per l'utente
2.2 I comandi dell'AmigaDOS per il programmatore
2.3 Tavola di consultazione rapida dei comandi dell'AmigaDOS

2.1 I comandi dell'AmigaDOS per l'utente

Terminologia non familiare

In questo manuale potreste trovare termini mai incontrati finora. La lista seguente comprende alcuni termini ricorrenti che possono confondere le idee se non sono familiari.

Boot sequenza di inizializzazione del sistema.

Codice oggetto output binario proveniente da un assembler o un compilatore e input binario di un linker.

Default impostazione iniziale o, in altre parole, quello che accade se non specificate nulla. In questo manuale il termine "default" è usato con il significato di "in assenza di altre specificazioni".

Device logico nome che può essere dato a una directory, per mezzo del comando ASSIGN, e si può poi usare come nome di device.

Disco sistema disco contenente il Workbench e i comandi.

Handle del file un valore usato internamente dall'AmigaDOS che rappresenta un file o un device aperto (utilizzato).

Nome del device parte di un nome che precede i due punti (:), per esempio CON:, df0:, PRT: e così via.

Nome di volume nome che viene dato a un disco fisico.

Reboot eseguire nuovamente un boot.

Stream un file o un device aperto associato a un handle. Per esempio, lo stream di input potrebbe venire da un file e lo stream di output potrebbe essere diretto verso il console device (CON:).

Nota: *il formato e il template dei comandi sono spiegati nella sezione 1.4.5, mentre nella sezione 1.7 sono illustrate le convenzioni adottate.*

;

Formato: [<comando>];[<commento>]

Template: "comando";"commento"

Scopo: aggiungere un commento alle linee dei comandi.

Spiegazione: il CLI ignora tutto ciò che è posto dopo il punto e virgola (;).

Esempi:

```
;Questa linea e' solo un commento
```

il CLI ignora la parte di linea contenente "Questa linea è solo un commento".

```
copy <file> to prt: ; stampa il file
```

copia il file sulla stampante, ma ignora il commento "stampa il file".

Vedere anche: EXECUTE

Formato: <comando>[>filename_di_output][<filename_di_input] [argomenti_comando*]

Template: "comando">"TO"<"FROM" "argomenti"

Scopo: dirigere l'input e l'output del comando.

Spiegazione: si usano i simboli > e < per ridirigere l'output e l'input di un comando. La direzione del vertice dei segni di maggiore e minore indica la direzione del flusso delle informazioni. Si possono usare questi simboli in ogni comando che riceve un input o fornisce un output. L'output di un comando viene generalmente visualizzato nella finestra corrente. Invece, se scrivete un simbolo > dopo un comando e prima di un filename, l'output verrà diretto verso questo file. Analogamente, se scrivete il simbolo < prima di un filename, il comando legge da questo file invece che dalla tastiera.

Non è necessario specificare sia la direzione TO (verso) sia la FROM (da), e neanche i file. L'esistenza e il numero degli "argomenti" dipende dal comando usato. La ridirezione avviene solo per il comando specificato. L'AmigaDOS ripristina l'output e l'input iniziali o di "default" (cioè la tastiera e la finestra corrente) non appena termina il comando. Occorre infine notare che la ridirezione deve *precedere* gli argomenti.

Esempi:

```
DATE > data_di_oggi
```

scrive l'output del comando DATE (che è la data di oggi e l'ora) nel file "data_di_oggi".

```
programma_mio < input_mio
```

comunica a programma_mio di accettare l'input dal file input_mio invece che dalla tastiera.

```
LIST > temp
SORT temp TO *
```

produce una lista dei file ordinata alfabeticamente e la visualizza sullo schermo.

La sequenza seguente:

```
ECHO > seconda_data 02-Jan-88
DATE < seconda_data ?
DELETE seconda_data
```

crea un file chiamato seconda_data che contiene il testo "02-Jan-88<line feed>". Successivamente utilizza tale file come input per il comando DATE. Notate che il "?" è necessario perché il DATE accetti l' input dal file anziché dalla linea di comando. Infine, visto che il file non serve più, il comando DELETE cancella seconda_data.

ADDBUFFERS

Formato: ADDBUFFERS df<x>: <nn>

Template: ADDBUFFERS "DRIVE/A,BUFFERS/A"

Scopo: ridurre il tempo di accesso al disco aggiungendo dei buffer-cache di settore.

Spiegazione: questo comando è disponibile solo a partire dalla versione 1.2 dell'AmigaDOS. Aggiunge <nn> buffer alla lista di cache dei settori per il drive <x>. Aggiungere buffer può ridurre sensibilmente il tempo di accesso al disco. Il prezzo da pagare è che il comando impegna più memoria: ogni buffer aggiunto riduce la memoria disponibile di circa 500 byte.

La riduzione del tempo di accesso al disco è meno rilevante se si aggiungono più di 25-30 buffer.

Esempi:

```
ADDBUFFERS df1:25
```

aggiunge 25 buffer ai cache di settore per il drive df1:. Se non si è sicuri riguardo al numero di buffer da aggiungere per la propria applicazione, questo è un valore iniziale ragionevole.

ASSIGN

Formato: ASSIGN [[<nome>]<dir>][LIST]

Template: ASSIGN "NAME,DIR,LIST/S"

Scopo: assegnare un nome di device logico a una directory del filing system.

Spiegazione: NAME è il nome del device logico da dare alla directory indicata da DIR. Se viene fornito solo NAME, l'AmigaDOS cancella il nome del device logico, cioè rimuove un'assegnazione precedente.

ASSIGN, senza parametri o con lo switch LIST, visualizza una lista delle assegnazioni correnti. Quando si usa ASSIGN, bisogna essere sicuri che ci sia un disco inserito nel drive. Questa verifica è importante poiché ASSIGN esegue un'assegnazione al volume del disco e non al drive.

Notate che gli effetti di ASSIGN vengono persi quando si esegue un "reboot" del computer.

Esempi:

```
ASSIGN sorgenti: :nuovo/lavoro
```

assegna il nome del device logico "sorgenti" alla directory ":nuovo/lavoro". Per avere accesso ai file in ":nuovo/lavoro", si può usare il nome del device logico "sorgenti", come in:

```
TYPE sorgenti:xyz
```

che visualizza il contenuto del file ":nuovo/lavoro/xyz".

```
ASSIGN LIST
```

visualizza i nomi dei device logici attualmente in uso. Quando viene eseguito il comando ASSIGN LIST, con la versione 1.2, vengono visualizzati i nomi completi delle directory.

BINDDRIVERS

Formato: BINDDRIVERS

Scopo: aggiunge i device di gestione dell'hardware addizionale.

Spiegazione: questo comando è disponibile solo a partire dalla versione 1.2. Il comando BINDDRIVERS normalmente fa parte del file di startup (Startup-Sequence). Viene usato per aggiungere i device che pilotano l'hardware addizionale, residenti nella directory SYS:Expansion, e che vengono configurati automaticamente attraverso la libreria "expansion". Per gli utenti, questo significa che, se ci sono delle icone per le espansioni hardware nel cassetto Expansion del Workbench, l'hardware viene configurato automaticamente quando si esegue il boot del sistema.

Esempi:

```
BINDDRIVERS
```

BREAK

Formato: BREAK <task>[ALL][C][D][E][F]

Template: BREAK "TASK/A,ALL/S,C/S,D/S,E/S,F/S"

Scopo: impostare i flag di attenzione in un dato processo.

Spiegazione: BREAK imposta i flag di attenzione specificati di un processo. C imposta il flag del CTRL-C, D quello del CTRL-D e così via. ALL imposta tutti i flag a partire dal CTRL-C fino al CTRL-F. Per default, l'AmigaDOS imposta solo il flag del CTRL-C. L'effetto del comando BREAK è identico a quello che si ottiene selezionando il processo in questione con il movimento del mouse sulla finestra, premendo il pulsante di selezione, e digitando la combinazione di tasti richiesta.

Esempi:

```
BREAK 7
```

imposta il flag di attenzione CTRL-C del processo 7. È la stessa cosa che selezionare il processo 7 e premere CTRL-C.

```
BREAK 5 D
```

imposta il flag di attenzione CTRL-D del processo 5.

```
BREAK 3 D E
```

imposta sia il CTRL-D sia il CTRL-E del processo 3.

CD

Formato: CD[<dir>]

Template: CD "DIR"

Scopo: impostare o cambiare la directory ed eventualmente il drive corrente.

Spiegazione: CD, senza parametri, visualizza il nome della directory corrente. Nel formato illustrato precedentemente, <dir> indica una nuova directory corrente. Se la directory che viene specificata non si trova nel drive corrente, allora il comando CD modifica anche quest'ultimo.

Per cambiare la directory corrente in quella che attualmente la contiene (sempre che esista), bisogna digitare CD seguito da una barra (/). In pratica CD / muove la directory corrente al livello gerarchico superiore finché questa non diventa la directory radice (cioè il più alto livello del filing system). È permesso l'uso di barre multiple; ogni barra (/) si riferisce al livello precedente.

Esempi:

```
CD dfl:lavoro
```

imposta come directory corrente "lavoro" residente nel disco "df1", e il drive corrente come "df1:".

```
CD SYS:com/basic
CD /
```

imposta come directory corrente "SYS:com".

CHANGETASKPRI

Formato: CHANGETASKPRI <priorità>

Template: CHANGETASKPRI "PRI/A"

Scopo: cambia la priorità dei task CLI.

Spiegazione: questo comando è disponibile solo a partire dalla versione 1.2 dell'AmigaDOS. L'Amiga impiega i numeri di priorità per determinare quale dei processi attivi al momento deve essere eseguito. Normalmente la maggior parte dei processi attivati dall'utente possiede una priorità 0, e il tempo e i cicli di istruzione della CPU vengono equamente suddivisi.

Il comando CHANGETASKPRI cambia la priorità del task CLI e di tutti quelli che in seguito saranno da esso attivati.

Benché i valori di priorità possano variare tra -128 e +127, bisogna specificare solamente valori tra -5 e +5 per evitare di distruggere importanti processi del sistema. Il comando non verifica il valore di <priorità>.

Esempi:

```
CHANGETASKPRI 5
```

il task CLI, e tutti gli altri processi attivati per suo tramite, avranno una priorità maggiore di qualunque altro task utente creato senza il comando CHANGETASKPRI.

COPY

Formato: COPY [FROM]<nome>[TO]<nome>[ALL][QUIET]

Template: COPY"FROM,TO/A,ALL/S,QUIET/S"

Scopo: copiare un file o una directory da un posto a un altro.

Spiegazione: COPY produce una copia del file o della directory nel file o nella directory specificata come TO. Il contenuto precedente del TO, se esistente, viene perduto.

Se si specifica il nome di una directory come FROM, COPY copia tutti i file in essa contenuti nella directory TO. Se non si indica una directory FROM, l'AmigaDOS utilizza la directory corrente. La directory TO *deve* esistere perché COPY funzioni correttamente; infatti se non esiste non viene creata dal comando.

Se viene specificato ALL, COPY copia anche i file contenuti in tutte le subdirectory. In questo caso crea automaticamente, se è necessario, le subdirectory nella directory TO. Il nome del file che viene copiato è visualizzato sullo schermo, a meno che non si usi lo switch QUIET.

Si può anche specificare la directory sorgente come un pattern. In questo caso, l'AmigaDOS copia tutti i file che verificano il pattern indicato. Consultare il comando LIST per una descrizione completa dei pattern. Si possono indicare livelli di directory così come pattern.

Poiché nella versione 1.2 del sistema operativo è stata migliorata l'organizzazione dei dati su disco, è possibile elevare le prestazioni del proprio Amiga copiando con il comando COPY ALL il contenuto di un disco in formato 1.1 in un altro formattato con la versione 1.2 .

Esempi:

```
COPY file1 TO :lavoro/file2
```

copia "file1" della directory corrente in "file2" della directory ":lavoro".

```
COPY TO df1:backup
```

copia tutti i file contenuti nella directory corrente in "df1:backup". Non copia nessuna subdirectory e df1:backup deve già esistere.

```
COPY df0: TO df1: ALL QUIET
```

esegue una copia logica del disco "df0:" su "df1:" senza visualizzare i nomi dei file.

```
COPY test.#? TO df1:xyz
```

copia tutti i file contenuti nella directory corrente che iniziano per "test." nella directory "xyz" del disco "df1:", assumendo che xyz già esista (per una descrizione dei pattern, come "#?", vedere il comando LIST in questo stesso capitolo).

```
COPY testo TO prt:
```

copia il file "testo" sulla stampante.

```
COPY * TO con:10/10/200/100/
```

Selezionate la finestra nella quale avete digitato il comando e scrivete qualcosa. Ogni volta che terminate una linea, questa viene visualizzata nella nuova finestra. Premete CTRL-\ quando avete terminato in modo da chiudere la finestra.

```
COPY df0:?/#? TO df1: ALL
```

copia ogni file contenuto in qualunque subdirectory, identificata da un nome che abbia solo un carattere, del disco df0: nella directory radice di df1:.

Vedere anche: JOIN

DATE

Formato: DATE [<data>][<ora>][TO|VER<nome>]

Template: DATE "DATE,TIME,TO=VER/K"

Scopo: visualizzare o impostare la data e l'ora di sistema.

Spiegazione: DATE senza parametri visualizza la data e l'ora attuali, incluso il giorno della settimana; l'ora viene mostrata adottando l'intero ciclo delle 24 ore.

DATE <data> imposta la data usando il formato GG-MMM-AA, dove MMM sono le prime tre lettere del nome del mese in inglese (per esempio il mese di gennaio è Jan). Se la data è già stata impostata, è possibile modificarla indicando il nome di un giorno in inglese (se si dichiara Tuesday la data viene spostata al martedì successivo) oppure le parole "tomorrow" e "yesterday" (rispettivamente "domani" e "ieri").

DATE <ora> imposta l'ora nel formato OO:MM (per Ore e Minuti). Bisogna aggiungere uno zero prima della cifra, quando si rende necessario. L'AmigaDOS, se si usano i due punti (:), riconosce che è stato indicato un orario invece che una data. Questo per dire che si possono impostare sia la data sia l'ora in qualsiasi ordine, visto che DATE si riferisce all'ora solo quando incontra la forma OO:MM.

Se non si imposta la data, il processo di convalida dei dischi l'aggiorna automaticamente, fissandola uguale a quella del file creato per ultimo. Riferitevi al capitolo 1 per i particolari di questa procedura.

Per specificare la destinazione della verifica, si usano le keyword TO e VER. La destinazione è solitamente il terminale, a meno che non venga dichiarato diversamente.

Nota: *se digitate DATE prima che il processo di convalida dei dischi sia terminato, l'ora apparirà non impostata; per aggiornarla potete usare DATE o semplicemente aspettare che il processo di convalida finisca.*

Esempi:

```
DATE
```

visualizza la data attuale.

```
DATE 06-Sep-88
```

imposta la data al 6 settembre 1988. L'orario non viene modificato.

```
DATE tomorrow
```

imposta la data al giorno dopo.

```
DATE TO giovanni
```

apre il file "giovanni", creandolo se inesistente, e vi scrive la data corrente.

```
DATE 10:50
```

aggiorna l'ora alle 11 meno 10.

```
DATE 23:00
```

aggiorna l'ora alle 11 di sera.

```
DATE 01-Jan-02
```

la data viene impostata al 1 gennaio 2002 (la più vecchia data a disposizione è l'1 gennaio 1978).

DELETE

Formato: DELETE <nome>[<nome>*][ALL][Q|QUIET]

Template: DELETE ",,,,,,,,,ALL/S,Q=QUIET/S"

Scopo: cancella fino a dieci file o directory.

Spiegazione: DELETE tenta di cancellare il primo file che viene indicato; se non vi riesce, lo schermo visualizza un messaggio e l'AmigaDOS ritenta con il file seguente nella lista. Una directory non può essere cancellata finché contiene almeno un file.

Si può anche usare un pattern per indicare i filename; riferitevi al comando LIST per avere una descrizione dettagliata della procedura. I pattern possono specificare livelli di directory così come nomi di file, e tutti i file che verificano il pattern vengono cancellati.

Se viene usato ALL in unione a un nome di directory, DELETE la cancellerà con tutte le subdirectory e i file che contiene.

A meno che non si indichi lo switch QUIET (oppure Q), quando viene cancellato un file, sullo schermo appare il suo nome.

Esempi:

```
DELETE vecchio
```

cancella il file "vecchio".

```
DELETE lavoro/prog1 lavoro/prog2 lavoro
```

cancella i file "prog1" e "prog2" della directory "lavoro", e poi cancella anche quella.

```
DELETE t#?/#?(1|2)
```

cancella tutti i file il cui nome termina con "1" o "2" e la cui directory ha un nome che inizia con "t" (per una spiegazione dei pattern si veda il comando LIST, più avanti in questo capitolo).

```
DELETE df1:#? ALL
```

cancella tutti i file in df1:

Vedere anche: DIR (opzione I-DEL)

DIR

Formato: DIR [<nome>][OPT A|I|AI]

Template: DIR "DIR,OPT/K"

Scopo: fornire una lista ordinata dei file contenuti in una directory. DIR può anche includere i file nelle subdirectory e può essere usato in modo interattivo.

Spiegazione: usato senza argomenti, DIR visualizza i file della directory corrente. Invece, seguito dal nome di una directory, fornisce i file della directory specificata. Il formato di visualizzazione usato è: prima le subdirectory, poi la lista ordinata dei file divisa in due colonne. Se si desidera assicurarsi che un file esiste, è possibile utilizzare il comando LIST filename.

La digitazione di DIR seguito da un filename, dove "filename" è il nome di un file che effettivamente esiste, causa soltanto la risposta: "filename is not a directory" (il filename non è una directory).

Per trasmettere un'opzione al comando DIR si usa la keyword OPT. È possibile indicare l'opzione A per includere nella lista ogni subdirectory che segua quella specificata. Ogni sublista appare rientrata.

Per visualizzare solo i nomi delle directory si deve usare l'opzione D.

L'opzione I comunica che DIR deve essere eseguito in modo interattivo. Ogni file, o directory, viene visualizzato con un punto di domanda. Per ottenere il nome successivo nella lista si deve premere RETURN. Q, invece, permette di uscire dal programma. Per tornare al livello di directory precedente o per fermarsi, se si è al livello della directory iniziale, si deve usare il tasto B.

Se il nome mostrato è quello di una directory, è possibile entrarvi per accedere a ulteriori file e subdirectory digitando E. Quindi, usando i tasti B ed E si possono selezionare i diversi livelli del filing system. Digitando il comando DEL (cioè premendo le tre lettere D E L, non il tasto DEL) si cancella una directory, sempreché sia rispettata la condizione che sia vuota.

Se il nome è invece quello di un file, DEL lo cancella mentre T lo visualizza sullo schermo. Nell'ultimo caso la pressione di CTRL-C ferma la visualizzazione e ritorna al modo di funzionamento interattivo.

Per attivare la funzione di aiuto circa le possibili risposte a una richiesta in modo interattivo, si può digitare ?.

Esempi:

```
DIR
```

fornisce una lista dei file che si trovano nella directory corrente. A partire dalla versione 1.2 del sistema operativo si può fermare l'output del comando DIR con CTRL-C.

```
DIR df0: OPT a
```

visualizza l'intera struttura di directory del disco df0:.

DISKCHANGE

Formato: DISKCHANGE <dr>

Template: DISKCHANGE "DEV/A"

Scopo: comunicare all'AmigaDOS che è stato cambiato un disco nel drive da 5,25".

Spiegazione: questo comando è disponibile solamente a partire dalla versione 1.2. Il comando DISKCHANGE informa l'AmigaDOS che è stato cambiato un disco nel drive <dr>. Questa operazione è necessaria per i drive da 5,25": si deve inserire il nuovo disco ed eseguire il comando

```
DISKCHANGE df2:
```

prima che l'AmigaDOS continui.

DISKCOPY

Formato: DISKCOPY [FROM]<disco>TO<disco>
[NAME<nome>]

Template: DISKCOPY "FROM/A,TO/A/K,NAME/K"

Scopo: copia il contenuto di un disco da 3,5" in un altro.

Spiegazione: DISKCOPY effettua una copia dell'intero contenuto del disco che si indica come FROM, scrivendo sopra all'eventuale contenuto del disco indicato come TO. DISKCOPY formatta automaticamente il nuovo disco mentre effettua la copia. Il comando viene normalmente usato per fare copie di backup dei dischi. Con la versione 1.2 il comando DISKCOPY funziona con qualunque disco (inclusi hard disk, partizioni di disco, e dischi da 5,25") che sia stato preventivamente configurato con il comando MOUNT, purché i dischi o le partizioni siano della stessa dimensione.

Una volta che il comando è stato impartito, l'AmigaDOS richiede che vengano inseriti i dischi appropriati; togliete il disco sistema e inserite il disco sorgente e quello destinazione secondo le indicazioni.

Il comando può essere usato per copiare qualunque disco da 3,5" in un altro, ma sorgente e destinazione devono avere la stessa struttura e dimensione. Per copiare informazioni tra dischi di dimensioni diverse, si deve fare uso del comando COPY.

Il comando può essere usato per effettuare copie di dischi anche se si dispone di un unico drive. Basta specificare sorgente e destinazione come lo stesso device, e il sistema manterrà in memoria tutte le informazioni del disco sorgente che può contenere. Poi richiederà l'introduzione del disco destinazione per scaricare il contenuto della memoria sul disco, e così via finché è necessario.

Se non viene indicato alcun nome, DISKCOPY crea il nuovo disco con lo stesso nome di quello vecchio. Comunque l'AmigaDOS riconosce la differenza tra due dischi che hanno lo stesso nome grazie alla data e all'ora di creazione associata a ciascuna directory radice. DISKCOPY assegna al nuovo disco come data e ora di creazione quella attuale del sistema.

Con la versione 1.2 può apparire per il comando DISKCOPY un requester del tipo seguente:

```
Diskcopy:
Failed to open icon.library

Retry                Cancel
```

Il requester appare quando DISKCOPY non riesce ad avere accesso alla icon library che viene tenuta nella directory libs del disco Workbench. Questo può accadere se avete digitato:

`COPY c:diskcopy to ram:`
(rimosso il disco Workbench)
`DISKCOPY df0: to df0:`
(appare un requester che richiede l'introduzione del disco contenente la directory LIBS:, normalmente il disco Workbench)

Se appare il requester, inserite il disco sistema (nota tecnica: quando questo capita, DISKCOPY ritenta la chiamata a OpenLibrary. Il flag per riprovare è DISKINSERTED).

Nota: *per copiare parte di un disco, si può usare il comando COPY selezionando RAM: come dispositivo destinazione.*

Esempi:

`DISKCOPY FROM df0: TO df1:`

effettua una copia di backup del disco df0: sul disco df1:.

`DISKCOPY FROM df0: TO df0:`

effettua una copia di backup del disco in df0: usando un solo drive.

Vedere anche: COPY

DISKDOCTOR

Formato: DISKDOCTOR <dr>:

Template: DISKDOCTOR "DRIVE/A"

Scopo: recuperare i dischi rovinati.

Spiegazione: questo comando è disponibile solo con la versione 1.2 del sistema operativo.
Se l'AmigaDOS riconosce un disco rovinato, visualizza un messaggio con il quale spiega che il disco non può essere convalidato (un disco può rovinarsi se, per esempio, viene estratto dal drive quando la spia è ancora accesa). DISKDOCTOR ricostruisce quanto più gli è possibile della struttura originaria del disco. Dopo aver eseguito il comando DISKDOCTOR, si devono copiare su un altro disco tutti i file che si vogliono conservare, e si deve procedere a una nuova formattazione del disco rovinato.

Esempi: se viene visualizzato un requester del tipo seguente, significa che l'AmigaDOS ha individuato qualche danno nel disco:

```
Volume
Testi
is not validated
```

oppure

```
Error validating disks
Disk is unreadable
```

Se il disco si trova nel drive df1:, digitate:

```
DISKDOCTOR df1:
```

Dopo che DISKDOCTOR ha terminato le sue operazioni, viene visualizzato il messaggio:

```
Now copy files required to a new disk and reformat this disk.
```

(Ora copiate i file necessari in un altro disco e formattate nuovamente questo)

ECHO

Formato: ECHO <stringa>

Template: ECHO " "

Scopo: visualizzare l'argomento dato.

Spiegazione: ECHO scrive il singolo argomento nello stream attuale di output, che può essere un file o un device. Normalmente è utile solo all'interno di una sequenza di comandi o come parte di un comando RUN. Se viene fornito un argomento in modo non corretto, viene visualizzato un messaggio di errore.

Esempi:

```
RUN COPY :lavoro/prog TO dfl:lavoro ALL QUIET +
ECHO ''Copia terminata''
```

crea tramite un processo in background un nuovo CLI per copiare la directory indicata. Quando ha terminato, lo schermo visualizza il messaggio:

```
Copia terminata
```

Se è stato creato il seguente file execute

```
ECHO ''Inizia il file execute MIACOPIA''
COPY dfl:ABC TO RAM:ABC
COPY dfl:XYZ TO RAM:XYZ
ECHO ''Estrai il disco da dfl:''
ECHO ''Inserisci il nuovo disco in dfl:''
WAIT 10 SECS
COPY RAM:ABC TO dfl:ABC
COPY RAM:XYZ TO dfl:XYZ
ECHO ''Fatto''
```

allora

```
EXECUTE MIACOPIA
```

copia i due file nel RAM disk e poi di nuovo su un disco.

ED

Formato: ED [FROM]<nome>[SIZE<n>]

Template: ED "FROM/A,SIZE"

Scopo: creare o modificare file di testo.

Spiegazione: ED è un editor di schermo e può essere impiegato in alternativa all'editor di linea EDIT. Il file indicato come FROM viene letto in memoria per essere modificato dai comandi di editing impostati dall'utente; se il file FROM non esiste, l'AmigaDOS ne crea uno nuovo.

Dal momento che il file viene letto in memoria, esiste un limite alle dimensioni del file che potete modificare per mezzo di ED. A meno che non sia indicato diversamente, la dimensione dell'area di lavoro è 40.000 byte, normalmente sufficiente per la maggior parte dei file. Comunque, per modificare le dimensioni dell'area di lavoro, è possibile indicare il nuovo numero di byte dopo la keyword SIZE.

Nel capitolo 3 è riportata una dettagliata descrizione del comando ED.

Esempi:

```
ED lavoro/prog
```

modifica il file "lavoro/prog" se esiste, altrimenti lo crea.

```
ED lunghissimo SIZE 50000
```

modifica il file "lunghissimo", allocando un'area di lavoro di 50.000 byte.

EDIT

Formato: EDIT [FROM]<nome>[[TO]<nome>][WITH<nome>] [VER<nome>][OPT<opzioni>]

Template: EDIT "FROM/A,TO,WITH/K,VER/K,OPT/K"

Scopo: modificare file di testo.

Spiegazione: EDIT è un editor di linea, cioè tratta un file sequenziale una linea alla volta. Se viene indicato TO, EDIT copia dal file FROM verso il file TO. Una volta terminate le operazioni di editing, il file TO contiene il risultato, mentre il contenuto di FROM non cambia. Se non viene specificato TO, allora EDIT scrive il risultato in un file temporaneo. Se date i comandi Q o W, EDIT fa assumere al file temporaneo il nome del file FROM, avendone però salvato in precedenza il contenuto nel file ":t/edit-backup". Se invece viene usato il comando STOP, EDIT non effettua nessun cambiamento nel file FROM.

EDIT legge i propri comandi dallo stream di input oppure dal file accompagnato a WITH, se è presente.

EDIT manda i propri messaggi e l'output di verifica al file specificato con la keyword VER, oppure allo schermo, se viene omesso.

OPT indica la presenza di opzioni: Pn imposta a "n" il massimo numero di linee precedenti, Wn la massima lunghezza di una linea. I valori iniziali sono P40W120.

Nota: *non si possono usare i simboli < e > per ridirigere l'input e l'output quando viene chiamato EDIT.*

Il capitolo 4 presenta una descrizione dettagliata del comando EDIT.

Esempi:

```
EDIT lavoro/prog
```

modifica il file "lavoro/prog". Quando le operazioni sono terminate, EDIT salva la vecchia versione di "lavoro/prog" in ":t/edit-backup".

```
EDIT lavoro/prog TO lavoro/nuovoprog
```

modifica il file "lavoro/prog" inserendo il risultato nel file "lavoro/nuovoprog".

```
EDIT lavoro/prog WITH edits/uno VER nil:
```

modifica il file "lavoro/prog" facendo uso dei comandi memorizzati nel file "edits/uno" e mandando i messaggi e l'output di verifica al device fittizio "nil:".

ENDCLI

Formato: ENDCLI

Template: ENDCLI

Scopo: terminare un processo CLI interattivo.

Spiegazione: l'AmigaDOS permette l'uso di ENDCLI, che rimuove il CLI selezionato dal mouse, solo come comando interattivo.

ENDCLI deve essere usato solo nei CLI creati per mezzo di un comando NEWCLI. Se il CLI iniziale (processo 1) viene fatto terminare, e nessun altro CLI è stato attivato usando il comando NEWCLI, l'effetto che si ottiene è quello di terminare la sessione d'uso dell'AmigaDOS.

Notate che il comando ENDCLI non prevede alcun argomento e che non esiste una verifica sulla validità o meno dei parametri.

Nota: *non fate esperimenti con ENDCLI senza aver prima usato NEWCLI: infatti l'uso di ENDCLI con il CLI iniziale vi tradisce sempre facendolo terminare. Se il CLI è stato attivato da Workbench non c'è problema, dal momento che vi si farà ritorno; se invece si aveva a disposizione solo quello, terminandolo non si avrà alcuna possibilità di crearne uno nuovo.*

Esempi: la sequenza seguente:

```
NEWCLI
LIST
ENDCLI
```

apre una nuova finestra, esegue una lista della directory e chiude la finestra CLI appena creata.

EXECUTE

Formato: EXECUTE <file_comando>[<arg>*]

Template: EXECUTE "file_comando","argomenti"

Scopo: eseguire un file di comandi con la sostituzione dei parametri.

Spiegazione: normalmente EXECUTE viene impiegato per risparmiare all'utente l'accesso alla tastiera. Il file_comando contiene una serie di comandi che vengono eseguiti dal CLI uno alla volta, proprio come se venissero introdotti per mezzo della tastiera. Se l'esecuzione del file crea una nuova finestra CLI, allora i risultati potrebbero essere diversi da quelli che si sarebbero ottenuti impartendo i comandi dalla tastiera. Quando viene usato EXECUTE nella versione 1.2, se non esiste già, viene creata la directory T:.

Si può usare il comando EXECUTE anche per effettuare la sostituzione di parametri (cioè variabili), in tutti i casi nei quali ha senso usare un nome come variabile. Prima che il file di comandi venga eseguito, l'AmigaDOS verifica che i nomi dei parametri combacino con quelli posti dopo il comando EXECUTE. I parametri possono avere valori specifici, ai quali l'AmigaDOS ricorre, quando altri valori non vengano esplicitamente indicati. Se un parametro non è stato specificato, e non esiste il corrispondente default, allora viene considerato vuoto e non viene operata nessuna sostituzione.

Per usare la sostituzione dei parametri, bisogna fornire al comando EXECUTE alcune direttive che, per essere indicate, devono appartenere a una riga che inizi con un punto (.). Le direttive disponibili sono le seguenti:

.KEY		Template degli argomenti, usata per specificare il formato degli argomenti, può venir abbreviata con .K
.DOT	car	Cambia il carattere DOT (inizialmente ".") con car
.BRA	car	Cambia il carattere BRA (inizialmente "<") con car
.KET	car	Cambia il carattere KET (inizialmente ">") con car

.DOLLAR	car	Cambia il carattere-default (inizialmente "$") con car, può venir abbreviato con .DOL
.DEF	valore keyword	Associa un default a un parametro
.<spazio>		Linea di commento
.<newline>		Linea di commento vuota

Prima dell'esecuzione, l'AmigaDOS scandisce il contenuto del file alla ricerca di ogni voce chiusa tra i caratteri BRA e KET (generalmente "<" e ">"). Una voce di questo tipo può consistere in una keyword oppure in una keyword e un valore di default da usare nel caso che la keyword non venga impostata (per separare la keyword dal default, se ce n'è uno, si deve usare il carattere DOLLAR, generalmente "$"). Così l'AmigaDOS rimpiazza <ANIMALE> con il valore che avete associato con la keyword ANIMALE, mentre rimpiazza <ANIMALE$CANE> con il valore di ANIMALE se c'è, altrimenti con il default rappresentato da CANE.

Un file può usare i "comandi punto" solo se la prima linea ha un punto come primo carattere. Il CLI analizza la prima linea e se questa è un comando punto, per esempio un commento (.<spazio>testo), allora scandisce l'intero file alla ricerca di eventuali sostituzioni di parametri e crea un file temporaneo nella directory :T. Se il file non inizia con un comando punto, si suppone che non ci siano comandi punto nel file e che quindi non si debba operare nessuna sostituzione di parametri. In questo secondo caso, il CLI inizia l'esecuzione del file direttamente senza copiarlo nella directory :T. Notate che si possono sempre aggiungere commenti nel file execute usando il carattere di commento del CLI: il punto e virgola (;). Se non c'è la necessità di sostituire dei parametri o di usare i comandi punto, si consiglia di non usarli in modo da ridurre gli accessi aggiuntivi al disco dovuti alla presenza del file temporaneo.

L'AmigaDOS fornisce una serie di comandi che si rivelano utili solo se usati in un file di comandi. Sono IF, SKIP, LAB, QUIT e possono essere nidificati.

Anche i file di comandi possono essere nidificati, cioè se ne può avere uno che contiene altri comandi EXECUTE.

Per fermare l'esecuzione di un file si deve premere CTRL-D. Se state eseguendo file di comandi nidificati, cioè se un file ne ha chiamato un altro, è possibile fermare l'intero insieme di file EXECUTE premendo CTRL-C; la pressione di CTRL-D ferma l'esecuzione solo del file corrente.

Esempi: supponete che il file "list" abbia il seguente contenuto:

```
.k filename/a
run copy <filename> to prt: +
echo ''Stampa di <filename> eseguita''
```

allora il comando

```
EXECUTE list test/prog
```

agirà come se fossero stati impartiti da tastiera i seguenti comandi:

```
RUN copy test/prog TO prt: +
ECHO ''Stampa di test/prog eseguita''
```

Un altro esempio "visualizza" alcune delle caratteristiche appena descritte:

```
.KEY nome/a
IF EXISTS <nome>
TYPE <nome> OPT n  (se il file si trova nella directory corrente
                    lo stampa con i numeri di linea)
ELSE
ECHO ''<nome> non si trova in questa directory''
ENDIF

RUN EXECUTE visualizza prog2
```

visualizzerà il file prog2 con i numeri di linea sullo schermo se si trova nella directory corrente, altrimenti apparirà il messaggio:

```
prog2 non si trova in questa directory
```

Vedere anche: ;, IF, SKIP, FAILAT, LAB, ECHO, RUN, QUIT

Altri esempi per il comando EXECUTE

Esempio 1: sostituzione di parametri per nome di keyword e/o posizione

L'istruzione .KEY (o .K) prevede l'identificazione dei parametri tramite le keyword o tramite la loro posizione nella linea; comunica a EXECUTE quanti parametri si deve aspettare e come li deve interpretare. In altre parole, .KEY serve come template per i valori dei parametri che si specificano. In un file di comandi è permessa una sola istruzione .KEY, che inoltre, se presente, deve essere nella prima linea.

Quando viene impartito un comando, l'AmigaDOS risolve la sostituzione dei parametri in due modi: per mezzo della presenza delle keyword prima dei parametri, o per la posizione relativa dei parametri nella linea di comando. La sostituzione per mezzo del nome della keyword ha la precedenza.

Supponiamo che il file chiamato DEMO1 contenga la seguente istruzione .KEY:

```
.KEY uno,due
```

Questo comunica all'AmigaDOS di attendersi due sostituzioni di parametri: <uno> e <due> (i simboli di maggiore e minore indicano il valore della keyword da sostituire al momento dell'esecuzione).

Digitando la seguente linea di comando:

```
EXECUTE DEMO1 due nome1 uno nome2
```

Il valore "nome2" viene assegnato a <uno>, mentre "nome1" a <due>.

Si possono omettere i nomi di keyword se la sostituzione dei parametri rispetta l'ordine dato dall'istruzione .KEY. Per esempio la riga che segue equivale a quella precedente:

```
EXECUTE DEMO1 nome2 nome1
```

Questo perché i valori corrispondono all'ordine delle keyword specificato nell'istruzione .KEY.

Si possono anche mischiare i due metodi per la sostituzione dei parametri. Supponiamo di avere un'istruzione .KEY con diversi parametri:

```
.KEY parola1,parola2,parola3,parola4
```

Il comando EXECUTE sostituisce i valori dei parametri preceduti dalle opportune keyword, quindi si occupa dei parametri rimasti in ordine di posizione.

Per esempio:

```
EXECUTE DEMO2 parola3 ccc parola1 aaa bbb ddd
```

Il parametro "ccc" viene assegnato a <parola3>, "aaa" viene assegnato a <parola1> e rimangono due parametri. Scandendo da sinistra a destra, l'istruzione .KEY trova che <parola2> non è stata ancora assegnata e quindi le associa la successiva voce di input: "bbb". Infine neanche <parola4> è stata assegnata e quindi prende l'ultima voce rimasta: "ddd".

Possono essere indicate condizioni speciali per la sostituzione dei parametri; ecco un esempio:

```
.KEY nome1/a,nome2/a,nome3,nome4/k
```

"/a" indica che deve essere fornito un valore per soddisfare i parametri nome1 e nome2. I valori per nome3 e nome4 sono invece opzionali, ma "/k" indica che <nome4>, se fornito, deve essere preceduto dalla keyword esplicita "nome4". Per esempio:

```
EXECUTE DEMO3 abc def ghi nome4 jkl
```

Se l'utente non fornisce i parametri richiesti (come nome1 e nome2 nell'esempio precedente), EXECUTE genera un messaggio d'errore.

Per fare un esempio che illustri l'uso dell'opzione /k, supponiamo di creare un file execute chiamato COMPILA che lasci opzionalmente indicare il filename verso il quale deve essere diretto l'output del compilatore. L'istruzione .KEY potrebbe essere:

```
.KEY cosacompila/a,fileoutput/k
```

Se l'utente inserisce un comando come:

```
EXECUTE COMPILA miofile FILEOUTPUT miooutput
```

il file comando dice che la keyword FILEOUTPUT è opzionale, ma se viene usata deve essere seguita da un valore. Quindi il comando precedente è corretto e l'output del compilatore viene mandato al file "miooutput".

Esempio 2: assegnazione dei parametri di default e di caratteri bracket differenti

La direttiva .DEF stabilisce un valore di default per una keyword, che viene preso in considerazione solo se l'utente non ha indicato alcun valore sulla

linea di comando. All'interno di un file di tipo execute è possibile determinare che un parametro non è stato inserito, confrontandolo con "" (due doppi apici in fila). Bisogna effettuare questo confronto prima di eseguire qualunque istruzione .DEF nel file comando.

Si possono assegnare i default in due modi distinti. Prima di tutto usando l'operatore $, ma in questo caso è necessario che il default venga specificato in ogni riferimento al parametro.

Per esempio nel comando seguente:

```
ECHO ''<parola1$defparola1> e' il default per PAROLA1''
```

"defparola1" è il default indicato per parola1 e viene stampato quando il precedente comando viene eseguito. Il secondo metodo è quello di definirlo globalmente. Per esempio, con l'assegnazione:

```
.DEF parola1 ''defparola1''
```

si può eseguire il comando:

```
ECHO ''<parola1> e' il default per PAROLA1''
```

se il parametro non è stato indicato come argomento del comando EXECUTE, l'output di entrambi i precedenti comandi ECHO sarà:

```
defparola1 e' il default per PAROLA1
```

Notate che l'impiego di un ulteriore .DEF per lo stesso parametro viene ignorato. Se infatti aggiungiamo:

```
.DEF parola1 ''Nuovo default''
ECHO ''<parola1> e' ANCORA il default per PAROLA1''
```

Il comando ECHO visualizzerà:

```
defparola1 e' ANCORA il default per PAROLA1
```

e il secondo assegnamento verrà ignorato.

Assegnazione di differenti caratteri bracket

Ogni volta che EXECUTE incontra i segni di minore e maggiore (caratteri bracket) vede se può sostituire un parametro al loro interno. Un parametro non fornito e senza default diventa una stringa nulla.

Supponete di voler usare una stringa contenente i caratteri di minore (<) e maggiore (>); si possono usare le direttive .BRA e .KET per definirne i sostituti. Per esempio,

```
ECHO ''Questa linea non stampa i segni < e >''
.BRA {
.KET }
ECHO ''Questa linea stampa i segni < e >''
ECHO ''Il default per parola1 e' {parola1}''
```

Il primo comando ECHO fa sì che EXECUTE ricerchi una sostituzione del parametro "e", visto che si trova all'interno dei caratteri < e >. Dal momento che "e" non è incluso nell'istruzione .KEY, allora gli viene sostituita la stringa nulla. Poi, dopo le direttive .BRA e .KET, che ridefiniscono i due caratteri in questione, il secondo ECHO stampa la linea

```
Questa linea stampa i segni < e >
```

Il terzo ECHO spiega che le parentesi graffe ora funzionano come simboli per la delimitazione delle keyword.

Esempio 3: simulazione della copia di file per mostrare la struttura dei file comando

L'istruzione IF permette di realizzare un confronto e, a seconda del risultato ottenuto, di far eseguire azioni diverse. Tra i confronti possibili vi sono quelli per determinare l'uguaglianza tra stringhe e quelli per verificare l'esistenza di un file. Si può usare un'istruzione ELSE in unione a IF per stabilire cosa bisogna fare nel caso che la condizione non risultasse vera. L'istruzione ELSE, se usata, viene considerata come parte integrante del blocco dell'IF che viene terminato per mezzo di ENDIF. I programmi di esempio che seguono fanno uso anche dell'istruzione SKIP. Questa permette di saltare SOLAMENTE IN AVANTI all'interno del file, fino a giungere a un'etichetta definita da un'istruzione LAB.

La struttura IF...ENDIF è illustrata nel breve esempio che segue. Generalmente è una buona pratica verificare che le keyword non siano state omesse o che non siano stringhe nulle, come riportato qui di seguito:

```
IF ''<parola1>'' EQ ''uso''
   SKIP USO
ENDIF
IF ''<parola2>'' EQ ''''
   SKIP USO
ENDIF
```

Racchiudendo le parole per la sostituzione dei parametri tra virgolette, all'interno di un'istruzione IF, si evita che EXECUTE generi un errore per la mancanza della keyword.

Se vengono omesse le virgolette oltre al valore, il risultato che ne deriva è una linea del tipo:

```
IF EQ ''uso''
```

che produce un errore a causa del fatto che IF e EQ sono adiacenti. Invece l'uso delle virgolette intorno agli indicatori di sostituzione della keyword genera una linea come:

```
IF '''' EQ ''uso''
```

che è ammessa.

Si può usare NOT in unione a IF per ribaltare il significato del test che viene compiuto. Per esempio:

```
IF NOT exists <file>
```

In una linea contenente IF non ci può essere altro che la condizione di test. Per esempio la linea seguente non è corretta:

```
IF <qualcosa> EQ vero SKIP fatto
```

La forma corretta dell'esempio precedente sarebbe:

```
IF <qualcosa> EQ vero
   SKIP fatto
ENDIF
```

Si noti che, se la costante stringa usata nel confronto avesse contenuto qualche spazio, sarebbe stato d'obbligo l'uso delle virgolette. In questo caso, invece, "VERO" sarebbe accettabile come VERO.

L'esempio successivo dimostra che gli IF possono venir nidificati in modo che i comandi possano essere eseguiti in base a condizioni multiple; notate poi che il comando EXECUTE permette la rientranza delle parole per rendere più leggibile un file con diversi livelli di nidificazione.

Il file comando che ora illustreremo simula un programma di copia di file mostrando alcune utili strutture dei file comando: IF...[ELSE]...ENDIF, LAB e SKIP.

```
.KEY da, verso, o/s                ;assegnazione lista dei parametri
IF ''<da>'' EQ ''''                ;verifica che venga fornito il file DA
```

```
    SKIP uso                                ;nessun file, mostra all'utente le
                                            ;istruzioni
ENDIF
IF ''<verso>'' EQ ''''                      ;verifica che venga fornito il file
                                            ;VERSO
    SKIP uso                                ;nessun file, mostra all'utente le
                                            ;istruzioni
ENDIF

IF NOT exists <da>                          ;verifica l'esistenza del file DA
    ECHO ''Il file sorgente non esiste.''
    ECHO ''Usate il comando DIR e riprovate.''
    SKIP fatto                              ;Nota: si può usare SKIP
                                            ;dall'interno di un IF
ENDIF

IF exists <verso>                           ;verifica l'esistenza del file VERSO
    IF ''<o>'' EQ ''O''                     ;l'utente ha indicato O come
                                            ;parametro?
        COPY FROM <da> TO <verso>
        ECHO ''Il file chiamato <verso> e' stato sostituito da una copia di <da>.''
        ECHO ''Richiesta soddisfatta.''
    ELSE
        ECHO ''Il comando sovrascrive un file esistente SOLO se''
        ECHO ''il parametro O viene specificato.''
        ECHO ''Richiesta negata.''
        SKIP uso                            ;spiega come si usa questo file
    ENDIF
ELSE
    COPY FROM <da> TO <verso>
ENDIF
SKIP fatto

LAB uso
ECHO ''Uso del file cp''
ECHO ''I seguenti formati di copia sono permessi:''
ECHO ''x cp DA file_sorgente VERSO file_destinazione''
ECHO ''x cp DA file_sorgente file_destinazione''
ECHO ''x cp file_sorgente VERSO file_destinazione''
ECHO ''x cp file_sorgente file_destinazione''
ECHO ''x cp VERSO file_destinazione DA file_sorgente''
ECHO ''x cp file_sorgente file_destinazione O''
ECHO ''x cp DA file_sorgente VERSO file_destinazione O''
ECHO ''x cp O DA file_sorgente VERSO file_destinazione''
```

```
ECHO ''dove: x e' l'abbreviazione per EXECUTE, cp e' il nome di ''
ECHO ''questo file comando, e ''O'' significa 'sovrascrivi il file esistente'.''

LAB FATTO
```

Esempio 4: esempio di un file batch iterativo

Il comando SKIP permette esclusivamente salti in avanti. Per creare una struttura iterativa in un file di comandi, si deve usare EXECUTE in modo ricorrente. Questo significa usare il comando EXECUTE all'interno del file stesso per inviare l'esecuzione a un'etichetta precedente. Il file che segue illustra un esempio di loop.

I cinque messaggi che seguono

```
Questo messaggio viene scritto una volta all'inizio (parm1,parm2).
I loop
II loop
III loop
Questo messaggio viene scritto una volta alla fine (parm1,parm2).
```

vengono visualizzati dal file di comandi:

```
.KEY parm1,parm2,loopcnt,looplabel
FAILAT 20
IF NOT ''<looplabel>'' EQ ''''          ;chiamata con etichetta?
   SKIP <looplabel>                      ;sì, allora itera
ENDIF

ECHO ''Questo messaggio viene scritto una volta all'inizio (<parm1>,<parm2>).''
LAB primo_loop
IF ''<loopcnt>'' EQ ''III''              ;finito di iterare?
   SKIP loopend_<looplabel>              ;sì, chiudi i loop
ENDIF
ECHO ''<loopcnt>I loop''
EXECUTE esempio_loop ''<parm1>'' ''<parm2>'' <loopcnt>I primo_loop

LAB loopend_<looplabel>
IF NOT ''<loopcnt>'' EQ ''''
   SKIP exit
ENDIF
                                         ; fine dei loop
ECHO ''Questo messaggio viene scritto una volta alla fine (<parm1>,<parm2>).''

LAB exit
```

FAILAT

Formato: FAILAT <n>

Template: FAILAT "RCLIM"

Scopo: comunicare a una sequenza di comandi di terminare la propria esecuzione se un comando dà luogo a un codice d'errore maggiore o uguale a "n".

Spiegazione: i comandi indicano che hanno fallito in qualche modo l'esecuzione impostando un codice d'errore. Un valore diverso da zero indica che il comando è incorso in un errore di qualche tipo. Un valore maggiore o uguale a un certo limite (chiamato limite di fallimento) fa terminare una sequenza di comandi non interattivi (cioè i comandi che vengono specificati dopo RUN oppure in un file per EXECUTE). Il codice d'errore indica quanto grave sia l'errore, e normalmente vale 5, 10 oppure 20.

Si può usare il comando FAILAT per modificare il limite di errore accettabile dal valore iniziale 10. Se si incrementa il livello, si specifica che una certa classe di errori non deve essere più considerata determinante e quindi l'esecuzione dei comandi successivi deve continuare dopo l'eventuale errore. L'argomento deve essere un numero positivo e il livello di errore accettabile viene posto nuovamente a 10 dopo il termine di una sequenza di comandi.

Si deve usare FAILAT prima di un comando come IF, che verifica se un comando ha generato un errore, altrimenti la sequenza termina prima dell'esecuzione dell'IF.

Se si omette l'argomento, viene visualizzato il valore attuale del limite di errore accettabile.

Esempi:

```
FAILAT 25
```

La sequenza di comandi termina solamente se un programma dà luogo a un codice d'errore maggiore o uguale a 25.

Vedere anche: IF, EXECUTE, RUN, QUIT

FAULT

Formato: FAULT [<n>*]

Template: FAULT ",,,,,,,,,"

Scopo: visualizzare il messaggio d'errore corrispondente al codice che viene fornito.

Spiegazione: l'AmigaDOS ricerca i numeri e visualizza i messaggi corrispondenti, fino a un massimo di dieci.

Esempi:

```
FAULT 222
```

visualizza il messaggio dell'errore 222.

```
FAULT 221 103 121 218
```

visualizza i messaggi degli errori 221, 103, 121 e 218.

FILENOTE

Formato: FILENOTE [FILE]<file>[COMMENT]<stringa>

Template: FILENOTE "FILE/A,COMMENT/A"

Scopo: associare un commento o una nota a un file.

Spiegazione: FILENOTE assegna un commento al file specificato.

La keyword COMMENT introduce un commento opzionale lungo fino a un massimo di 80 caratteri e contenente anche più di una parola (che presenti cioè spazi tra i caratteri). Nel caso si usino più parole bisogna racchiudere il commento tra virgolette (").

Un commento viene associato a un file particolare e viene visualizzato nella linea sotto il file quando lo si esamina per mezzo del comando LIST, come appare qui di seguito:

```
prog      30 rwed Today 11:07:33
: version 3.2  27-Mar-87
```

Quando viene creato un file, normalmente con esso non viene generato alcun commento. Se un file dotato di commento viene sovrascritto da un altro file, viene mantenuto il commento anche se il contenuto è stato alterato. Se un file dotato di commento viene copiato per mezzo del comando COPY, perde il commento, nel senso che il file destinazione non possiede più il commento di quello originale, ma può invece avere, se viene copiato in un file già esistente, quello del file cancellato dalla sovrascrittura.

Esempi:

```
FILENOTE prog2 COMMENT ''Version 3.2  27-Mar-87''
```

associa il commento "Version 3.2 27-Mar-87" al file "prog2".

Vedere anche: LIST

FORMAT

Formato: FORMAT DRIVE <nome_drive> NAME <stringa>

Template: FORMAT "DRIVE/A/K,NAME/A/K"

Scopo: formattare e inizializzare un nuovo disco da 3,5''.

Spiegazione: il comando formatta un floppy disk nuovo nella maniera richiesta dall'AmigaDOS. Una volta che il disco è stato formattato, viene inizializzato e gli viene associato il nome che è stato indicato. Notate che bisogna usare assolutamente entrambe le keyword DRIVE e NAME. Le uniche opzioni che possono venir fornite dopo la keyword DRIVE sono df0:, df1:, df2: e df3:. Dopo NAME, può essere indicata qualunque stringa, ma se si usano degli spazi bisogna racchiuderla fra virgolette ('').

ATTENZIONE: FORMAT formatta e inizializza un disco facendolo diventare un disco vuoto, quindi se usate un disco non vuoto perderete il suo precedente contenuto.

Il nome assegnato al disco deve essere unico; può essere lungo da uno a trenta caratteri e può essere composto da più parole separate da spazi. Ma se è presente qualche spazio, il nome deve essere racchiuso fra virgolette ('').

FORMAT, con la versione 1.2 del sistema operativo, agisce su qualunque disco (inclusi dischi rigidi, partizioni di dischi e floppy da 5,25'') che sia stato precedentemente installato con il comando MOUNT. Inoltre la sintassi del comando FORMAT è stata modificata in:

FORMAT DRIVE <nome_drive>
NAME <nome_disco> [NOICONS]

Se non includete l'opzione NOICONS, viene aggiunta un'icona che rappresenta il Trashcan (il cestino). Il comando può essere interrotto prima della sua conclusione premendo CTRL-C.

Quando viene usato da CLI, FORMAT scandisce nuovamente il cammino delle directory di comandi, se la ricerca

finale fallisce. Questo accade perché potrebbe verificarsi il caso che l'utente abbia inserito, come risultato di un requester, un disco, rendendo così validi uno o più percorsi (path); così il percorso viene ripetuto due volte prima che il comando fallisca.

Nota: *non è necessario formattare un disco se si ha intenzione di usarlo come destinazione di un comando DISKCOPY.*

Esempi:

```
FORMAT DRIVE df0: NAME ''disco lavoro''
```

formatta e inizializza il disco nel drive df0: con il nome "disco lavoro". Il comando INITIALIZE è una nuova opzione, presente con la versione 1.2, del menu del Workbench, e analogamente al comando FORMAT, agisce su ogni disco installato con il comando MOUNT.

Quando si usa INITIALIZE viene aggiunta nel disco l'icona di un Trashcan. Se avete il vostro disco SYS: in un drive, il disco userà l'immagine del Trashcan di quel disco per il nuovo Trashcan. Altrimenti sarà aggiunta la nuova immagine del Trashcan che appare sui nuovi dischi del Workbench. Per arrestare l'inizializzazione di un disco prima che INITIALIZE abbia finito, si deve premere CTRL-C.

Vedere anche: DISKCOPY, INSTALL, RELABEL

IF

Formato: IF [NOT][WARN][ERROR][FAIL][<stringa> EQ <stringa>][EXISTS <nome>]

Template: IF "NOT/S,WARN/S,ERROR/S,FAIL/S,EQ/K,EXISTS/K"

Scopo: permettere la verifica di condizioni all'interno di sequenze di comandi.

Spiegazione: IF può essere usato solo in un file di comandi utilizzato da EXECUTE. Se una o più delle condizioni specificate vengono soddisfatte, IF esegue tutti i comandi seguenti finché non incontra un comando ENDIF oppure un ELSE; altrimenti esegue tutti i comandi che seguono ELSE, se quest'ultimo esiste, (ENDIF e ELSE sono utili solo in sequenze contenenti IF). ENDIF termina un comando IF, mentre ELSE fornisce un'alternativa se le condizioni di IF non vengono soddisfatte. Notate che le condizioni e i comandi relativi a un comando IF oppure a un ELSE possono estendersi su più di una riga prima dei corrispondenti ENDIF.

La tavola seguente illustra alcune possibili forme in cui usare IF, ELSE e ENDIF:

IF <condizione>	IF <condizione>	IF <condizione>
<comandi>	<comandi>	<comandi>
ENDIF	ELSE	IF <condizione>
	<comandi>	<comandi>
	ENDIF	ENDIF
		ENDIF

Notate che ELSE è opzionale e che gli IF nidificati saltano all'ENDIF più vicino.

L'opzione ERROR è disponibile solo se FAILAT è stato usato per impostare un limite di fallimento maggiore di 10. Similmente, FAIL è disponibile se il livello è stato impostato a più di 20.

Keyword	**Funzione**
NOT	inverte il risultato
WARN	soddisfatta se il codice di ritorno precedente è ≥ 5.
ERROR	soddisfatta se il codice di ritorno precedente è ≥ 10.

FAIL	soddisfatta se il codice di ritorno precedente è ≥20.
<a> EQ <b>	soddisfatta se le stringhe "a" e "b" sono identiche (non hanno importanza il maiuscolo e il minuscolo).
EXISTS <file>	soddisfatta se il file esiste.

Si può usare IF EQ per scoprire se un parametro non è stato impostato in un file di comandi usando la forma:

```
IF <a> EQ ''''
```

Esempi:

```
IF EXISTS lavoro/prog
TYPE lavoro/prog
ELSE
ECHO ''File non trovato''
ENDIF
```

Se il file "lavoro/prog" esiste, viene visualizzato dall'AmigaDOS; altrimenti viene mostrato il messaggio "File non trovato" e l'esecuzione prosegue con il successivo comando della sequenza.

```
IF ERROR
SKIP lab_errore
ENDIF
```

Se il comando precedente si è interrotto generando un codice di ritorno ≥10, allora l'AmigaDOS salta tutti i comandi della sequenza finché non incontra l'etichetta "lab_errore" definita per mezzo di un comando LAB.

```
IF ERROR
IF EXISTS marco
ECHO ''Il file 'marco' esiste, ma si e' verificato ugualmente un
errore.''
ENDIF
ENDIF
```

Vedere anche: FAILAT, SKIP, LAB, EXECUTE, QUIT

INFO

Formato: INFO

Template: INFO

Scopo: fornire informazioni riguardo al filing system.

Spiegazione: il comando visualizza una serie di informazioni su ogni unità a dischi: queste informazioni sono la massima dimensione del disco, la parte di disco usata e quella libera, il numero di errori software presenti, lo stato del disco e il nome.

Esempi:

```
INFO
Unit   Size   Used   Free   Full   Errs  Status      Name
df1:   880K      2   1756     0%      0  Read/Write  Test-6
df0:   880K   1081    677    61%      0  Read/Write  AmigaDOS CLI

Volumes available:

Test-6 [Mounted]
AmigaDOS CLI [Mounted]
```

INSTALL

Formato: INSTALL [DRIVE]<drive>

Template: INSTALL "DRIVE/A"

Scopo: installare la parte del sistema operativo che rende un disco in grado di effettuare un boot.

Spiegazione: lo scopo del comando INSTALL è di mettere un disco in condizione di rispondere positivamente a un boot (cioè si usa INSTALL per creare un disco in grado di inizializzare l'Amiga). Per far questo è sufficiente indicare il nome del drive nel quale è inserito il disco che deve diventare uno "startup disk". Ci sono quattro possibili nomi di drive: df0:, df1:, df2: e df3:.

Esempi:

```
INSTALL df0:
```

fa sì che il disco inserito nel drive "df0:" sia in grado di effettuare un boot.

JOIN

Formato: JOIN <nome><nome>[<nome>*]AS<nome>

Template: JOIN ",,,,,,,,,,,,,,,AS/A/K"

Scopo: concatenare fino a 15 file per formarne uno nuovo.

Spiegazione: l'AmigaDOS copia i file indicati, nell'ordine dato, in un nuovo file che non può avere lo stesso nome di un file di input.

Esempi:

```
JOIN parte1 parte2 AS versione_finale
```

unisce insieme i due file, ponendo il risultato in "versione_finale". I due file originali rimangono inalterati, mentre "versione_finale" contiene una copia di "parte1" e "parte2".

LAB

Formato: LAB <testo>

Template: LAB "testo"

Scopo: dichiarare un'etichetta nei file comando.

Spiegazione: il comando ignora qualunque parametro venga fornito. LAB dev'essere usato per definire un'etichetta "testo" che viene cercata dal comando SKIP.

Esempi:

```
LAB errore
```

definisce l'etichetta "errore" alla quale SKIP può saltare.

Vedere anche: SKIP, IF, EXECUTE

LIST

Formato: LIST [[DIR]<dir>][P|PAT<pat>][KEYS] [DATES][NODATES][TO<nome>][S<str>] [SINCE<data>][UPTO<data>][QUICK]]

Template: LIST "DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K,UPTO/K,QUICK/S"

Scopo: esaminare e visualizzare informazioni specifiche riguardo a una directory o un file.

Spiegazione: se non si indica un nome (il parametro DIR), LIST visualizza il contenuto della directory corrente. Il primo parametro accettato da LIST è DIR. Si hanno a disposizione tre opzioni. DIR può essere un filename e in questo caso LIST visualizza le informazioni per quest'unico file. DIR può essere un nome di directory e quindi LIST mostra le informazioni per ogni file o directory che essa contiene. Per ultimo, se viene omesso il parametro DIR, LIST si riferisce alla directory corrente e ne visualizza il contenuto corredato di informazioni (per i dettagli riguardo alla directory corrente riferirsi al comando CD).

Nota: *LIST, al contrario di DIR, non ordina il contenuto della directory prima di visualizzarlo.*

Se non è accompagnato da altre opzioni, LIST mostra:

```
filename   dimensione protezione data ora
:commento
```

Questi campi sono definiti come segue:

filename:
nome del file o della directory.

dimensione:
la dimensione del file in byte. Se il file non contiene nulla, questo campo indica "empty", che significa vuoto. Per le directory è invece "dir".

protezione:
specifica quale accesso è permesso per questo file; "rwed" indica Read, Write, Execute e Delete (cioè che sono possibili le operazioni di lettura, scrittura, esecuzione e cancellazione).

data e ora:
la data e l'ora della creazione del file.

commento:
questo è il commento associato usando il comando FILENOTE. Notate che è preceduto dai due punti (:).

Opzioni disponibili:

TO
Indica il file o il device verso il quale deve essere diretto l'output. Se viene omesso, l'output è indirizzato verso la finestra CLI corrente.

KEYS
Visualizza il numero del blocco di ogni header di file o directory.

DATES
Visualizza le date nel formato GG-MMM-AA (di default a meno che non si usi QUICK).

NODATES
Non mostra le informazioni riguardanti la data e l'ora.

SINCE <data>
Vengono visualizzati solo i file che sono stati modificati in data uguale o successiva a <data>. <data> deve essere nella forma GG-MMM-AA (con MMM in inglese) oppure può essere il nome di un giorno dell'ultima settimana (per esempio MONDAY) o TODAY (oggi) o YESTERDAY (ieri).

UPTO <data>
Analoga all'opzione precedente solo che i file selezionati devono essere precedenti alla <data> specificata.

P <pat>
Ricerca i file che combaciano con <pat>.

S <str>
Ricerca i file che contengono nel loro nome la stringa <str>.

QUICK
Visualizza solo i nomi dei file e delle directory (come il comando DIR).

È possibile indicare il range di ricerca dei file visualizzati in due modi. Il più semplice è quello di usare la keyword S, che riduce i file solo a quelli che contengono nel proprio nome la stringa specificata. Per indicare una regola di ricerca più dettagliata, si deve usare la keyword P o PAT seguita da un "pattern" corrispondente, come spiegato di seguito.

Un pattern consiste in un certo numero di caratteri speciali dotati di significato particolare.

I caratteri speciali sono: '()?%#|

Per cancellare gli effetti speciali di questi caratteri, bisogna che siano preceduti da un apostrofo ('); così '? corrisponde al carattere ? mentre '' corrisponde a '.

<table>
<tr><td>?</td><td>Corrisponde a qualunque carattere singolo.</td></tr>
<tr><td>%</td><td>Corrisponde alla stringa nulla.</td></tr>
<tr><td>#<p></td><td>Corrisponde a zero o più ricorrenze del pattern <p>.</td></tr>
<tr><td><p1><p2></td><td>Corrisponde a una sequenza composta dal pattern <p1> seguito dal pattern <p2>.</td></tr>
<tr><td><p1>|<p2></td><td>Corrisponde all'uno o all'altro pattern.</td></tr>
<tr><td>()</td><td>Raggruppa insieme i pattern. In mancanza di parentesi <p>, <p1> e <p2> rappresentano un singolo carattere.</td></tr>
</table>

Così:

<table>
<tr><td>LIST PAT A#BC</td><td>Corrisponde a AC, ABC, ABBC e così via.</td></tr>
<tr><td>LIST PAT A#(B|C)D</td><td>Corrisponde a AD, ABD, ABCD e così via.</td></tr>
<tr><td>LIST PAT A?B</td><td>Corrisponde a AAB, ABB, ACB e così via.</td></tr>
<tr><td>LIST PAT A#?B</td><td>Corrisponde a AB, AXXB, AZXQB e così via.</td></tr>
<tr><td>LIST PAT '?#?'#</td><td>Corrisponde a ?#, ?AB#, ??## e così via.</td></tr>
<tr><td>LIST PAT A(B|%)#C</td><td>Corrisponde a A, ABC, ACCC e così via.</td></tr>
<tr><td>LIST PAT #(AB)</td><td>Corrisponde a AB, ABAB, ABABAB e così via.</td></tr>
</table>

Esempi:

```
LIST
```

visualizza le informazioni riguardanti tutti i file e le directory contenute nella directory corrente, per esempio:

```
File_1
File 2
File.3
:commento
File004
```

(notate che "File.3" ha un commento)

```
LIST lavoro S nuovo
```

visualizza le informazioni riguardanti i file contenuti nella directory "lavoro" e il cui nome contiene il testo "nuovo". Notate che LIST S produce la risposta: "Args no good for key" (argomenti non corretti). Questo accade perché il comando LIST riconosce S come opzione, e non come argomento. Per trasformare una delle possibili opzioni in argomento, è necessario racchiuderla fra virgolette (''). Per esempio, LIST "S" visualizza il contenuto della directory s.

```
LIST lavoro P nuovo#?(x|y)
```

esamina la directory "lavoro", mostra le informazioni riguardanti tutti i file i cui nomi cominciano con "nuovo" e finiscono con "x" oppure "y".

```
LIST QUICK TO file1
```

invia al file di output "file1" solo i nomi, uno per ogni linea, dei file contenuti nella directory corrente. Sarà poi possibile modificare il file inserendo TYPE all'inizio di ogni riga ed eseguire il comando:

```
EXECUTE file1
```

per vedere il contenuto di ogni file.

Vedere anche: DATE, DIR, FILENOTE, PROTECT

MAKEDIR

Formato: MAKEDIR <dir>

Template: MAKEDIR "/A"

Scopo: creare una nuova directory.

Spiegazione: MAKEDIR crea una directory con il nome che viene indicato. Il comando crea una sola directory alla volta, così quelle che eventualmente si trovano lungo il percorso devono già esistere. Il comando fallisce se nella directory che la precede in gerarchia si trova già un file o una directory con lo stesso nome.

Esempi:

```
MAKEDIR test
```

crea la directory "test" all'interno di quella corrente.

```
MAKEDIR df1:xyz
```

crea la directory "xyz" all'interno della radice del disco "df1:".

```
MAKEDIR df1:xyz/abc
```

crea la directory "abc" all'interno della "xyz", che deve già esistere, del disco "df1:".

Vedere anche: DELETE

MOUNT

Formato: MOUNT <dr>:

Template: MOUNT "DEVICE/A"

Scopo: rendere disponibile un nuovo device (come un disk drive).

Spiegazione: questo comando è disponibile solo per la versione 1.2 del sistema operativo. Per usare un nuovo device con l'Amiga, bisogna prima crearne la corrispondente entry nel file "devs:Mountlist". Poi si può eseguire il comando MOUNT. Di solito i comandi MOUNT sono inclusi nel file comando "s/Startup-Sequence", in modo da rendere disponibili i nuovi device durante l'attivazione del sistema.

Normalmente i produttori di hardware non forniscono le procedure necessarie per installare i loro device. Per questo motivo il file "devs:Mountlist" contiene delle entry d'esempio che potete modificare per i vostri device.

Il formato per un mountlist che supporti un drive da 5,25", come terzo drive, è

```
DF2:            Device = trackdisk.device
                Unit = 2
                Flags = 1
                Surface = 2
                BlocksPerTrack = 11
                Reserved = 2
                PreAlloc = 11
                Interleave = 0
                Lowcyl = 0; Highcyl = 39
                Buffers = 5
                BufMemType = 3
```

Si può includere su una stessa linea più di una descrizione, separandola per mezzo del punto e virgola (;). I numeri sono considerati decimali, a meno che non vengano preceduti da "0x" nel qual caso vengono considerati come esadecimali. I commenti possono essere posti nel file nello stesso modo usato per i programmi in C.

Se il device non è del tipo che memorizza informazioni, si può usare questo formato per l'entry:

```
AUX:            Handler = L:aux-handler
                Stacksize = 700
                Priority = 5
```

NEWCLI

Formato: NEWCLI [<finestra>]

Template: NEWCLI "WINDOW"

Scopo: creare una finestra associata al nuovo processo CLI interattivo.

Spiegazione: l'AmigaDOS crea una nuova finestra CLI che diventa quella dell'attuale ambiente di lavoro. La nuova finestra ha impostate come directory corrente e stringa di prompt le stesse di quella da cui è stato eseguito NEWCLI. Ogni finestra CLI è indipendente, e permette quindi operazioni di input, di output e l'esecuzione di programmi separati.

Per collegare la tastiera con la nuova finestra occorre puntare il mouse al suo interno e premere il tasto sinistro (cioè il bottone di selezione). Per selezionare un nuovo CLI potete utilizzare qualunque posizione all'interno della finestra .

Quando viene eseguito NEWCLI senza argomenti, l'AmigaDOS crea una finestra di posizione e dimensioni standard. Per cambiare la dimensione di una finestra, bisogna muovere il mouse in modo che il cursore punti sull'angolo inferiore destro (gadget di dimensionamento), premere il tasto di selezione e spostare il mouse, mantenendo premuto il tasto, finché non si raggiunge la dimensione voluta. Per cambiare la posizione della finestra bisogna muovere il mouse sulla barra di spostamento (quella dove si trova il titolo), premere il pulsante sinistro e muoverlo dove si desidera posizionare la finestra.

Per personalizzare una finestra CLI, è possibile indicarne la posizione esatta, la dimensione ed eventualmente un titolo. Per farlo, la sintassi da usare è la seguente:

```
CON:x/y/larghezza/altezza/titolo
```

dove "CON:" denota una finestra console, "x" e "y" sono le coordinate della posizione della finestra, "larghezza" e "altezza" sono le dimensioni della finestra e "titolo" è la stringa da porre nella barra di spostamento. Il titolo è opzionale, ma è necessaria la presenza della barra finale (/). Tutte le dimensioni sono espresse in pixel di schermo.

Esempi:

```
NEWCLI
```

crea un nuovo processo CLI e lo attiva come CLI corrente.

```
NEWCLI CON:10/30/300/100/mioCLI
```

crea un nuovo CLI alla posizione (10,30), della dimensione di 300x100 pixel e con l'intestazione "mioCLI".

```
NEWCLI ''CON:20/15/300/100/il mio CLI''
```

Le virgolette permettono l'uso di titoli con spazi tra le parole. Per ulteriori informazioni sul console device (CON:) riferirsi alla sezione 1.3.6, "Comprensione dei nomi dei device".

Con la versione 1.2 è stata introdotta una nuova opzione FROM per il comando NEWCLI. Il formato per NEWCLI FROM è:

NEWCLI [FROM <filename>]

L'inclusione dell'opzione FROM <filename> obbliga il CLI che viene creato a eseguire i comandi nel file di nome <filename>.

Nota: *a differenza di un processo in background creato con il comando RUN, un processo NEWCLI si pone in stato d'attesa dopo che è stato creato.*

Vedere anche: ENDCLI, RUN

NOTEPAD

Formato: NOTEPAD [<nome-file>] [-q]

Template: NOTEPAD "NOME-FILE,-Q"

Scopo: attivare l'applicazione Notepad.

Spiegazione: con la versione 1.2 è possibile mandare in esecuzione l'applicazione Notepad da CLI, anche se non rientra nei comandi dell'AmigaDOS. Indicando il nome di un file, Notepad lo elabora automaticamente. L'opzione "-q" informa l'applicazione che non è necessario ricercare tutte le fonti carattere disponibili; in questo caso viene caricata in memoria solo la fonte impiegata nel file già elaborato precedentemente. Se Notepad non trova il file indicato, non provvede a cercarlo fino a quando l'utente non lo richiede espressamente.

Esempi:

```
NOTEPAD
```

manda in esecuzione l'applicazione Notepad.

```
NOTEPAD df1:memo
```

manda in esecuzione l'applicazione, e questa provvede a cercare il file "memo" nella directory radice del disco nel drive df1:.

```
RUN NOTEPAD df1:utilities/memo -q
```

l'applicazione Notepad viene mandata in esecuzione come un processo. L'opzione "-q" indica a Notepad che non deve ricercare tutte le fonti carattere disponibili.

Vedere anche: RUN

PATH

Formato: PATH [SHOW]|[[ADD]<dir>[<dir>]...]
|[RESET[<dir>][<dir>]...]

Template: PATH ",,,,,,,,,,ADD/S,SHOW/S,RESET/S"

Scopo: aggiungere, vedere o cambiare le directory nelle quali l'AmigaDOS ricerca i comandi da eseguire.

Spiegazione: questo comando è disponibile dalla versione 1.2 del sistema operativo.

Il comando PATH permette di vedere, aggiungere o cambiare i "percorsi di ricerca" (search paths): le directory nelle quali l'AmigaDOS ricerca i comandi da mandare in esecuzione. Per default l'AmigaDOS ricerca i programmi (comandi) prima nella directory corrente, e se non li trova accede alla directory C: (solitamente SYS:C). Il comando PATH sprovvisto di parametri, o seguito dalla keyword SHOW, visualizza i percorsi di ricerca correntemente impostati. PATH, seguito da una o più directory, aggiunge ai percorsi di ricerca già impostati quelli delle directory indicate; oppure è possibile far seguire a PATH la keyword ADD prima delle directory da aggiungere, senza alterare il risultato dell'operazione.

Con un comando PATH [ADD] è possibile aggiungere fino a dieci directory, i cui nomi devono essere separati almeno da uno spazio.

È possibile ripristinare i percorsi di ricerca di default del sistema (directory corrente e directory C:) impartendo il comando PATH RESET. In questo modo tutti i percorsi che sono stati aggiunti a quelli di sistema vengono cancellati. PATH RESET seguito dai nomi di una o più directory rende attivi i nuovi percorsi.

Le keyword possono essere posizionate sia all'inizio che alla fine della linea di comando. Per esempio entrambe le forme PATH <dir1> <dir2> RESET e PATH RESET <dir1> <dir2> sono valide.

Esempi: `PATH SHOW` (oppure `PATH`)

visualizza le directory nelle quali l'AmigaDOS ricerca i comandi. Alcuni tipici percorsi possono essere:

```
Current directory
Workbench:System
C:
```

`PATH ADD sys:nuovicomandi` (oppure `PATH sys:nuovicomandi`)

L'AmigaDOS ricercherà i comandi anche nella directory sys:nuovicomandi (per default, nella versione 1.2 il file s/Startup-Sequence include un comando PATH ADD sys:System).

```
Current directory
Workbench:System
Workbench:nuovicomandi
C:
```

Potreste voler aggiungere questo comando nel vostro file s/Startup-Sequence in modo che l'AmigaDOS ricerchi automaticamente anche nella directory "sys:nuovicomandi" ogni volta che viene usato questo disco. Se guardate nel file s/Startup-Sequence vedrete che è in questo modo che SYS:System è stato aggiunto ai percorsi di ricerca.

PROMPT

Formato: PROMPT <str>

Template: PROMPT "PROMPT"

Scopo: cambiare il prompt del CLI corrente.

Spiegazione: se non viene fornito alcun parametro, l'AmigaDOS imposta il prompt di default (">"). Altrimenti il prompt diventa quello indicato in <str>. L'AmigaDOS può accettare anche una combinazione speciale di caratteri (%N) come viene spiegato nell'esempio seguente.

Esempi:

```
PROMPT
```

imposta come prompt ">".

```
PROMPT ''%N>''
```

imposta come prompt "n>", dove n è il numero del processo CLI corrente, dal momento che l'AmigaDOS interpreta la combinazione %N come il numero del processo.

PROTECT

Formato: PROTECT [FILE]<filename>[FLAGS<stato>]

Template: PROTECT "FILE,FLAGS/K"

Scopo: impostare lo stato di protezione di un file.

Spiegazione: PROTECT accede a un file e ne imposta lo stato di protezione.

La keyword FLAGS ha quattro opzioni: lettura (r), scrittura (w), cancellazione (d), esecuzione (e). Per indicare queste opzioni bisogna aggiungere i caratteri r, w, e oppure d dopo il nome del file. Se una di queste opzioni viene omessa, PROTECT presume che non sia richiesta; per esempio se vengono fornite tutte le opzioni tranne d, PROTECT garantisce che il file non può essere cancellato. Le opzioni di lettura, scrittura e cancellazione si possono riferire a qualunque tipo di file. L'AmigaDOS, nella versione attuale, presta attenzione solo al flag di cancellazione (d); comunque, gli utenti e i loro programmi possono, se lo desiderano, impostare e controllare gli altri flag.

Il comando PROTECT, con la versione 1.2, è stato modificato in modo che alteri solo i quattro bit meno significativi del campo di protezione e preservi gli altri, mentre precedentemente li poneva tutti a 1. Il bit 4 del campo di protezione, l'archive bit, viene ora posto a zero ogni volta che un file nel quale si è scritto qualcosa viene chiuso oppure una directory viene modificata. Questa caratteristica permette a un programma d'archiviazione di scorrere un disco per cercare i file che sono stati modificati o aggiunti dall'ultima volta che è stato eseguito, facendo anche in modo che la funzione PROTECT imposti a 1 l'archive bit per segnalare che il file è stato archiviato.

Esempi:

```
PROTECT prog1 FLAGS r
```

imposta lo stato di protezione del file "prog1" come "sola lettura".

```
PROTECT prog2 FLAGS rwd
```

il file "prog2" può essere letto, scritto e cancellato.

Vedere anche: LIST

QUIT

Formato: QUIT [<codice_di_ritorno>]

Template: QUIT "RC"

Scopo: uscire da una sequenza di comandi con un determinato codice d'errore.

Spiegazione: QUIT legge il file comandi dall'inizio alla fine e poi si ferma restituendo un codice di ritorno, che per default è zero.

Esempi:

```
QUIT
```

esce dalla sequenza di comandi.

```
FAILAT 30
IF ERROR
QUIT 20
ENDIF
```

Se l'ultimo comando ha generato un errore, viene terminata la sequenza di comandi con un codice di ritorno pari a 20. Per ulteriori informazioni sulle sequenze di comandi, si consiglia di studiare la spiegazione del comando EXECUTE.

Vedere anche: EXECUTE, IF, LAB, SKIP

RELABEL

Formato: RELABEL [DRIVE]<drive>[NAME]<nome>

Template: RELABEL "DRIVE/A,NAME/A"

Scopo: cambiare il nome di volume di un disco.

Spiegazione: RELABEL cambia il nome di volume di un disco assegnandogli il <nome> che è stato indicato. I nomi di volume vengono inizialmente assegnati durante la formattazione di un disco.

Esempi:

```
RELABEL df1: ''Disco con un nome nuovo''
```

Vedere anche: FORMAT

RENAME

Formato: RENAME [FROM]<nome>[TO|AS]<nome>

Template: RENAME "FROM/A,TO=AS/A"

Scopo: cambiare il nome a un file o a una directory.

Spiegazione: RENAME cambia il nome del file FROM mettendogli quello specificato con TO. FROM e TO devono essere filename dello stesso disco. Il nome FROM si può riferire a un file o a una directory. Se il filename si riferisce a una directory, RENAME ne lascia inalterato il contenuto (cioè le directory e i file all'interno di questa directory mantengono lo stesso nome e lo stesso contenuto).

Usando il comando RENAME viene cambiato solo il nome della directory. Se viene cambiato il nome a una directory o se viene usato RENAME per porre un file in un'altra directory (per esempio, RENAME :marco/lettera :roberta/lettera), l'AmigaDOS ne cambia la posizione nella gerarchia del filing system. Usare RENAME è come cambiare il nome di un file e poi spostarlo in un'altra directory. Altri sistemi operativi descrivono questa operazione come il "movimento" di un file o di una directory.

Il comando RENAME non viene eseguito se l'unico cambiamento richiesto è il passaggio di uno o più caratteri da maiuscoli a minuscoli o viceversa. Per esempio

```
RENAME volpe TO Volpe
```

non viene eseguito.

Nota: *se esiste già un file con lo stesso nome del file TO, RENAME non viene eseguito evitando che un file venga sovrascritto per sbaglio.*

Esempi:

```
RENAME lavoro/progl AS :arturo/esempio
```

cambia il nome del file "lavoro/prog" in ":arturo/esempio".

La directory radice deve contenere la directory "arturo" affinché il comando funzioni.

RUN

Formato: RUN <comando>

Template: RUN comando + comando ...

Scopo: eseguire i comandi come processi in background.

Spiegazione: RUN crea un processo CLI non interattivo e gli cede il resto della propria linea di comando come input. Il CLI in background esegue i comandi e si cancella automaticamente.

Il nuovo CLI possiede la stessa directory corrente e la stessa dimensione dello stack dei comandi del CLI dal quale RUN è stato impartito.

Per separare i comandi bisogna aggiungere un segno di addizione (+) e premere RETURN. RUN interpreta la linea che segue un + (RETURN) come la continuazione della linea precedente. Così si può costituire un'unica linea di comando composta da diverse linee fisiche facendole terminare tutte con un segno di addizione (+). Per mandare in esecuzione l'intera struttura RUN, è necessario premere il RETURN al termine di una riga senza farlo precedere dal carattere di addizione (+).

RUN visualizza il numero del processo appena creato e, nella versione 1.2, non cambia più la priorità del CLI da cui è stato eseguito.

Esempi:

```
RUN COPY :t/es1 PRT: +
DELETE :t/es1 +
ECHO ''Stampa terminata''
```

stampa il file "es1" copiandolo verso il device della stampante, lo cancella e visualizza il messaggio finale.

```
RUN EXECUTE seqcom
```

esegue in background tutti i comandi contenuti nel file "seqcom".

SEARCH

Formato: SEARCH [FROM]<nome>|<pat> [SEARCH]<stringa>[ALL]

Template: SEARCH "FROM,SEARCH/A,ALL/S"

Scopo: cercare una stringa di testo nei file contenuti in una directory.

Spiegazione: SEARCH svolge la ricerca attraverso tutti i file della directory specificata e, se viene indicata la keyword ALL, anche in tutte le sue subdirectory. SEARCH visualizza tutte le linee contenenti la <stringa> specificata come SEARCH e il nome del file cercato. Si può anche sostituire la directory FROM con un pattern (vedere il comando LIST per una completa descrizione dei pattern). Se si fa uso di un pattern, SEARCH effettua la propria ricerca solo nei file il cui nome combacia con il pattern specificato.

L'AmigaDOS ricerca sia il maiuscolo sia il minuscolo della stringa SEARCH. Notate che si deve racchiudere la stringa tra virgolette se contiene degli spazi.

Come al solito, per abbandonare il comando si deve premere CTRL-C. Per abbandonare la ricerca nel file attuale e proseguire con il successivo occorre invece premere CTRL-D.

Esempi:

```
SEARCH SEARCH uno
```

ricerca nei file della directory corrente la presenza della stringa "uno".

```
SEARCH df0: ''Ciao a tutti'' ALL
```

ricerca la stringa "Ciao a tutti" in ogni file del disco "df0:".

```
SEARCH test-#? uno
```

ricerca il testo "uno" in tutti i file della directory corrente il cui nome inizia per "test-".

SETDATE

Formato: SETDATE <file><data>[<ora>]

Template: SETDATE "FILE/A,DATE/A,TIME"

Scopo: cambiare la data associata a un file o una directory.

Spiegazione: questo comando, che è disponibile solo a partire dalla versione 1.2 del sistema operativo, permette di cambiare la data associata a un file o a una directory. Il formato della data e dell'ora sono gli stessi adottati per il comando DATE.

SETMAP

Formato: SETMAP <nome_mappa>

Scopo: cambiare la "mappa" della tastiera in quella adottata da un altro Paese.

Spiegazione: questo comando è disponibile con la versione 1.2 del sistema operativo.

Esiste una serie di mappe che descrivono le tastiere usate in Paesi diversi dagli Stati Uniti.

La directory "DEVS:keymaps" contiene i nomi dei file che descrivono le mappe di tasti disponibili nel disco Workbench. Per selezionare la mappa italiana, per esempio, digitate:

```
SETMAP i
```

Per tornare alla mappa di default memorizzata nella memoria a sola lettura, digitate:

```
SETMAP usa
```

È possibile modificare il proprio file "s/Startup-Sequence" in modo che includa un comando SETMAP.

SKIP

Formato: SKIP <etichetta>

Template: SKIP "LABEL"

Scopo: effettuare un salto in una sequenza di comandi.

Spiegazione: SKIP si usa in unione a LAB (riferirsi al comando LAB per ulteriori particolari). SKIP legge il file di comandi dalla linea successiva fino alla fine, cercando un'etichetta definita con LAB e senza eseguire alcun comando.

Si può usare SKIP con o senza l'indicazione dell'etichetta: senza indicazione SKIP ricerca il successivo comando LAB che non abbia un nome, mentre quando l'indicazione esiste, cerca il LAB che abbia l'etichetta indicata. LAB deve essere la prima voce presente in una linea del file. Se SKIP non trova l'etichetta specificata, l'AmigaDOS termina la sequenza e mostra il messaggio:

```
label ''<etichetta>'' not found by Skip
```

(Skip non ha trovato l'etichetta "<etichetta>")

Il comando SKIP può effettuare solamente salti in avanti all'interno del file di comandi.

Esempi:

```
SKIP
```

salta al successivo comando LAB che non ha un nome definito.

```
IF ERROR
SKIP errore
ENDIF
```

Se il comando precedente si è fermato per un errore, SKIP ricerca l'etichetta "errore" presente successivamente nel file di comandi.

```
FAILAT 100
ASSEM testo
IF ERROR
SKIP errore
ENDIF
```

```
ALINK
SKIP fatto
LAB errore
ECHO ''Errore in fase di assemblaggio''
LAB fatto
ECHO ''Prossimo comando, per piacere''
```

Vedere anche: EXECUTE, LAB, IF, FAILAT, QUIT

SORT

Formato: SORT [FROM]<nome>[[TO]<nome>]
[COLSTART<n>]

Template: SORT "FROM/A,TO/A,COLSTART/K"

Scopo: ordinare alfabeticamente e numericamente semplici file.

Spiegazione: questo comando è un programma di ordinamento molto semplice. SORT può essere usato per ordinare i file anche se non è sufficientemente veloce per i file di grosse dimensioni e non può operare se non c'è abbastanza memoria per caricare l'intero file.

FROM specifica il file sorgente, mentre l'output ordinato viene mandato al file TO. SORT presume che FROM sia un normale file di testo dove ogni linea è separata con un CR (Carriage Return). Ogni linea del file viene ordinata alfabeticamente in ordine crescente senza distinzione tra caratteri maiuscoli e minuscoli.

Per alterare, seppure in modo limitato, questa modalità di ordinamento si può usare la keyword COLSTART per specificare la prima colonna da cui deve iniziare il confronto. SORT poi ordina i caratteri della linea dalla posizione indicata in poi; se una linea è uguale a un'altra anche dopo quest'operazione, vengono usati per il confronto anche i caratteri dall'inizio della linea fino alla colonna che precede quella specificata da COLSTART.

Nota: *la dimensione iniziale dello stack (che è di 4000 byte) è sufficiente solo per piccoli file, che contano meno di 200 linee. Se si desidera ordinare file più grandi bisogna usare il comando STACK per incrementare la dimensione dello stack; per indovinare di quanto dev'essere aumentata la dimensione dello stack si deve far conto un po' sull'intuito e un po' sull'esperienza.*

ATTENZIONE: se lo stack è troppo piccolo l'Amiga va in crash. Se non siete sicuri è meglio indicare una dimensione molto superiore alle vostre esigenze.

Esempi:

```
SORT testo TO testo-ordinato
```

ordina alfabeticamente ogni linea di informazioni in "testo" e pone il risultato in "testo-ordinato".

```
SORT indice TO indice-ordinato COLSTART 4
```

ordina il file "indice", nel quale ogni record contiene il numero di pagina nelle prime tre colonne e la voce indice nel resto della linea, mettendo il risultato nel file "indice-ordinato". Il file di output è ordinato secondo l'indice, e nel caso di indici uguali, secondo il numero di pagina.

Vedere anche: >, <, STACK

STACK

Formato: STACK [<n>]

Template: STACK "SIZE"

Scopo: visualizzare o impostare la dimensione dello stack per i comandi.

Spiegazione: un programma, quando viene eseguito, usa una certa quantità di spazio dello stack. Nella maggior parte dei casi la dimensione iniziale dello stack, 4000 byte, è sufficiente, ma può essere cambiata per mezzo del comando STACK. Per fare questo, bisogna introdurre il comando STACK seguito dal nuovo valore dello stack, specificato in byte. STACK usato da solo visualizza la dimensione attuale dello stack.

L'unico comando per il quale normalmente è necessario cambiare la dimensione dello stack è SORT. I comandi ricorrenti come DIR hanno bisogno di uno stack di maggiori dimensioni solo se vengono usati con una struttura di directory profonda più di sei livelli.

> **ATTENZIONE:** se andrete oltre le disponibilità dello stack riceverete un unico avviso, e cioè che l'Amiga andrà in crash! Se non siete sicuri, è meglio indicare una dimensione molto superiore alle vostre esigenze.

Esempi:

```
STACK
```

visualizza la dimensione attuale dello stack.

```
STACK 8000
```

imposta lo stack a 8000 byte.

Vedere anche: RUN, SORT

STATUS

Formato: STATUS [[PROCESS]<processo>][FULL][TCB][SEGS] [CLI|ALL]

Template: STATUS "PROCESS,FULL/S,TCB/S,SEGS/S,CLI=ALL/S"

Scopo: visualizzare le informazioni riguardanti i processi CLI attualmente esistenti.

Spiegazione: STATUS usato da solo visualizza in una lista i numeri dei processi CLI e i programmi in esecuzione in ognuno di essi.

<processo> indica un numero di processo, e le informazioni si riferiscono solo a questo; se viene omesso, le informazioni sono mostrate per tutti i processi.

FULL = SEGS + TCB + CLI

SEGS visualizza i nomi delle sezioni della segment list di ogni processo.

TCB si riferisce alle informazioni riguardanti la priorità, la dimensione dello stack e quella del vettore globale di ogni processo.

Per ulteriori dettagli sullo stack e sulla grandezza del vettore globale, potete riferirvi al *Manuale dell'AmigaDOS di riferimento tecnico* in questo stesso volume.

CLI identifica i processi della Command Line Interface e mostra il nome di sezione del comando, o dei comandi, attualmente caricato, se esiste.

Esempi:

```
STATUS
```

visualizza brevi informazioni riguardanti tutti i processi.

```
STATUS 4 FULL
```

visualizza una completa descrizione del processo numero 4.

TYPE

Formato: TYPE [FROM]<nome>[[TO]<nome>][OPT N|H]

Template: TYPE "FROM/A,TO,OPT/K"

Scopo: visualizzare un file di testo oppure scriverne il contenuto in forma esadecimale.

Spiegazione: TO indica il file di output; se viene omesso, l'output viene mandato allo stream di output attuale, e questo significa, nella maggior parte dei casi, che viene mandato alla finestra corrente.

I tab (caratteri di tabulazione) contenuti nel file vengono espansi dal console device, e non dal comando TYPE. Per interrompere l'output occorre premere CTRL-C, mentre per sospenderlo è sufficiente premere un qualunque tasto (in genere la barra spaziatrice). Per riprendere l'output si preme RETURN oppure CTRL-X.

OPT specifica un'opzione di TYPE. La prima è "N" e causa l'introduzione dei numeri di linea nell'output.

La seconda opzione di TYPE è "H": ogni word del file viene visualizzata come numero esadecimale.

Esempi:

```
TYPE lavoro/prog
```

visualizza il file "lavoro/prog".

```
TYPE lavoro/prog OPT n
```

visualizza il file "lavoro/prog" con i numeri di linea.

```
TYPE obj/prog OPT h
```

visualizza il codice contenuto in "obj/prog" in formato esadecimale.

WAIT

Formato: WAIT <n>[SEC|SECS][MIN|MINS][UNTIL<ora>]

Template: WAIT ",SEC=SECS/S,MIN=MINS/S,UNTIL/K"

Scopo: attendere che trascorra un determinato periodo di tempo.

Spiegazione: è possibile usare WAIT nelle sequenze di comandi o dopo un comando RUN, se si desidera far trascorrere un certo periodo di tempo oppure aspettare una certa ora del giorno. A meno che non venga stabilito diversamente, il tempo di attesa è in secondi.

Il parametro indica il numero di secondi (o minuti se viene fornito MINS) che devono passare.

La keyword UNTIL permette di attendere una specifica ora del giorno, data nel formato OO:MM.

Esempi:

```
WAIT
```

attende un secondo.

```
WAIT 10 MINS
```

attende 10 minuti.

```
WAIT UNTIL 21:15
```

attende fino alle nove e un quarto di sera.

WHY

Formato: WHY

Template: WHY

Scopo: spiegare perché il comando precedente è fallito.

Spiegazione: di solito quando un comando fallisce viene mostrato un breve messaggio, il quale indica che qualcosa non è andato come previsto. Nella maggior parte dei casi il messaggio include il nome del file (se questo è il problema) senza scendere in ulteriori dettagli. Per esempio il comando

```
COPY * TO xyz
```

potrebbe fallire e dare origine al messaggio:

`Can't open xyz` (Non si può aprire xyz)

Questo potrebbe dipendere da una varietà di ragioni; per esempio "xyz" potrebbe essere già il nome di una directory, o potrebbe non esserci spazio a sufficienza sul disco per aprire i file o, infine, il disco potrebbe essere a sola lettura. COPY non fa differenza tra queste cause perché in genere l'utente conosce il motivo dell'errore. Comunque, immediatamente dopo che un comando è fallito potete digitare WHY (in inglese significa perché) seguito da RETURN per ottenere un messaggio che descriva in dettaglio che cosa non ha funzionato.

Esempi:

```
TYPE df0:
Can't open df0:
```
(Non si può aprire df0:)

```
WHY
Last command failed because object not of required type
```
(L'ultimo comando è fallito perché l'oggetto specificato non era del tipo richiesto)

WHY fornisce un'indicazione del perché il comando è fallito. "TYPE df0:" è fallito perché l'AmigaDOS non permette di stampare un device.

Vedere anche: FAULT

2.2 I comandi dell'AmigaDOS per il programmatore

I comandi illustrati in questo capitolo erano originariamente contenuti nei primi dischi del Workbench distribuiti ai programmatori. Dopo quella prima produzione, alcune software house hanno iniziato a commercializzare applicazioni in grado di svolgere le stesse funzioni con maggiore flessibilità, e questi comandi non sono più stati inseriti nel Workbench. Sono stati ugualmente riportati in questo manuale per motivi di completezza. In particolare, diversi pacchetti di programmazione oggi in circolazione mantengono ancora le stesse caratteristiche dei comandi ALINK e ASSEM.

Per quanto invece riguarda DOWNLOAD e READ, oggi esistono in commercio diversi pacchetti di comunicazione in grado di svolgere molte più funzioni di quelle offerte da questi due comandi.

ALINK

Formato: ALINK [[FROM|ROOT]<filename>
[,<filename>*|+<filename>*]][TO<nome>]
[WITH<nome>][VER<nome>][LIBRARY|LIB<nome>]
[MAP<mappa>][XREF<nome>][WIDTH<n>]

Template: ALINK "FROM=ROOT,TO/K,WITH/K,VER/K,LIBRARY
=LIB/K,MAP/K,XREF/K,WIDTH/K"

Scopo: raggruppare insieme varie sezioni di codice in un unico file eseguibile.

Spiegazione: ALINK viene usato per raggruppare insieme i file oggetto, generando un file eseguibile. Inoltre gestisce automaticamente i riferimenti alle librerie e costruisce i file di overlay. L'output di ALINK è rappresentato da un file caricabile dal Loader ed eseguibile sotto il controllo del supervisore per gli overlay, se necessario.

Per i dettagli e una completa spiegazione del comando ALINK, riferitevi al capitolo 4 del *Manuale dell'AmigaDOS per il programmatore*.

Esempi:

```
ALINK a+b+c TO output
```

collega i file "a", "b" e "c" producendo in output un unico file.

ASSEM

Formato: ASSEM [PROG|FROM]<prog>[-O<codice>] [-V<verso>][-L<list>][-E][-C|OPT<opt>] [-I<lista_dir>]

Template: ASSEM "PROG=FROM/A,-O/K,-V/K,-L/K,-H/K,-E/K,-C =OPT/K,-I/K"

Scopo: assemblare un programma nel linguaggio assembly del microprocessore MC68000.

Spiegazione: ASSEM assembla un programma nel linguaggio assembly del MC68000. Per ulteriori particolari, vedere il capitolo 3 del *Manuale dell'AmigaDOS per il programmatore*, in questo stesso volume.

PROG	è il file sorgente.
-O	è il file oggetto (cioè l'output binario dell'assemblatore).
-V	è il file verso il quale vengono inviati i messaggi che normalmente sono visualizzati sullo schermo.
-L	è il file nel quale viene scritto il listato del programma.
-C	specifica le opzioni dell'assemblatore.
-H	è un file d'intestazione che viene inserito all'inizio del file sorgente, come se si fosse usata una direttiva INCLUDE al suo interno.
-I	specifica una lista di directory nelle quali cercare i file da includere.
-E	è il file che riceve le direttive di assegnazione EQU contenute nel file sorgente. Si usa -E per generare un file di intestazione contenente tali direttive.

Le opzioni che si possono specificare con OPT oppure -C sono le seguenti:

S	produce una copia della tavola dei simboli come parte del file oggetto.
X	produce un file di cross-reference (riferimento incrociato).
W<n>	imposta la dimensione dell'area di lavoro a "n" byte.

Esempi:

```
ASSEM prog.asm TO prog.obj
```

assembla il programma sorgente "prog.asm" mettendo il risultato nel file "prog.obj". Scrive qualunque errore sullo schermo, ma non produce alcun file di list.

```
ASSEM prog.asm TO prog.obj -H slib -L prog.list
```

assembla il programma precedente con lo stesso output, ma include il file "slib" e pone il listing nel file "prog.list".

```
ASSEM xyz.asm -O xyz.obj OPT W8000
```

assembla un programma *molto* piccolo.

DOWNLOAD

Template: DOWNLOAD "FROM/A,TO/A"

Scopo: trasferire programmi nell'Amiga.

Spiegazione: il comando DOWNLOAD permette di trasferire i programmi scritti con un altro computer (per esempio un Sun) nell'Amiga.

Per usare DOWNLOAD bisogna possedere un BillBoard. Per trasferire il file caricabile dal Sun all'Amiga bisogna digitare sul Sun:

```
binload -p &
```

(questo bisogna farlo una volta sola), poi occorre digitare sull'Amiga:

```
download <filename Sun> <filename Amiga>
```

Prima di inizializzare il proprio Sun, bisogna assicurarsi che sia il BillBoard sia l'Amiga siano già accesi e funzionanti, altrimenti non vengono riconosciuti dal Sun. Il <filename Sun> dovrebbe terminare per convenzione con .ld. Una volta che si è trasferito il file è sufficiente, per eseguirlo, digitare <filename Amiga>.

Occorre notare che il comando "binload" non è un comando dell'AmigaDOS. Binload è usato sul Sun per caricare in formato binario i file che devono essere trasferiti al proprio Amiga.

DOWNLOAD accede sempre ai file del Sun relativi alla directory dalla quale è stato eseguito binload. Se non ci si ricorda la directory dalla quale è stato eseguito binload, bisogna specificare l'intero nome. Per fermare l'esecuzione di binload sul Sun, bisogna impartire un "ps" e poi un "kill" con il PID. Un reset software del computer comunica a binload di scrivere un messaggio nel proprio output standard (il default è la finestra dalla quale è stato eseguito). Se DOWNLOAD si arresta, premete CTRL-C per eliminarlo.

Il capitolo 1 del *Manuale dell'AmigaDOS per il programmatore* descrive in dettaglio come trasferire i programmi tra un PC IBM e l'Amiga, tra il Sun e l'Amiga e fornisce anche alcuni suggerimenti su come operare i trasferimenti da altri computer.

Esempi:

```
binload -p &
```
(sul Sun)

```
download test.ld test
```
(sull'Amiga)

oppure

```
download /usr/fred/DOS/test.ld test
```
(sull'Amiga)

poi digitate "test".

READ

Template: READ "TO/A,SERIAL/S"

Scopo: leggere dati dalla porta parallela o dalla porta seriale e memorizzarli in un file.

Spiegazione: il comando READ si pone in attesa che sulla porta parallela giungano una serie di caratteri esadecimali. Se si usa la keyword SERIAL, READ attende i dati sulla porta seriale. Ogni coppia di cifre esadecimali viene posta in memoria sotto forma di byte. READ, che riconosce Q come terminatore della sequenza di dati, ignora gli spazi, i caratteri newline e i tab. Riconosce invece le cifre ASCII 0-9 e le lettere maiuscole da A a F. Bisogna trasmettere una cifra esadecimale ASCII per ogni nibble (4 bit) e avere un numero pari di nibble. Quando la sequenza di dati è completa, READ trasferisce i byte dalla memoria nel file che è stato specificato.

Nota: *questo comando può essere usato per trasferire file binari o di testo.*

> **ATTENZIONE 1:** bisogna fare attenzione nel leggere due volte (con READ) lo stesso file; infatti READ, usato una seconda volta, cancella il contenuto precedente.
>
> **ATTENZIONE 2:** si possono perdere alcuni caratteri usando velocità di trasmissione elevate con la porta seriale.

Esempi:

```
READ TO df0:nuovo
```

Legge il file "df0:nuovo" dalla porta parallela.

```
READ nuovo SERIAL
```

Legge il file "nuovo" dalla porta seriale.

2.3 *Tavola di consultazione rapida dei comandi dell'AmigaDOS*

Comandi per l'utente

Utility per i file

;	carattere per i commenti.
><	dirigono, rispettivamente, l'input e l'output.
ADDBUFFERS	riduce il tempo d'accesso ai dischi aggiungendo un certo numero di buffer.
BINDDRIVERS	installa i driver per le periferiche addizionali.
CHANGETASKPRI	altera la priorità di un task CLI.
COPY	copia un file in un altro oppure copia tutti i file di una directory in un'altra.
DELETE	cancella fino a dieci file o directory.
DIR	visualizza i nomi dei file di una directory.
DISKCHANGE	comunica all'AmigaDOS che è stato cambiato un disco nel drive da 5,25".
DISKDOCTOR	cerca di recuperare un disco danneggiato.
ED	mette a disposizione un editor per i file di testo.
EDIT	mette a disposizione un editor di linea.
FILENOTE	aggiunge a un file una nota lunga al massimo 80 caratteri.
JOIN	concatena fino a 15 file per formarne uno nuovo.
LIST	esamina e visualizza informazioni dettagliate riguardanti un file o a una directory.
MAKEDIR	crea una nuova directory con il nome indicato.
NOTEPAD	manda in esecuzione il programma Notepad da CLI.
PATH	aggiunge, vede o cambia i percorsi di ricerca dell'AmigaDOS.
PROTECT	imposta lo stato di protezione di un file.
RENAME	cambia nome a un file o a una directory.
SEARCH	cerca una stringa di testo specificata all'interno di tutti i file di una directory.
SETDATE	cambia la data associata a un file o a una directory.
SETMAP	cambia la mappa della tastiera in quella del Paese desiderato.
SORT	ordina file semplici.
TYPE	mostra il contenuto di un file in modo testo o esadecimale.

Controllo del CLI

BREAK	imposta i flag di attenzione di un determinato processo.
CD	imposta la directory e/o il drive corrente.
ENDCLI	termina un processo CLI interattivo.
NEWCLI	crea un nuovo processo CLI interattivo.
PROMPT	modifica il prompt nel CLI corrente.
RUN	esegue i comandi come processi in background (sottofondo).
STACK	mostra o imposta la dimensione dello stack per i comandi.
STATUS	mostra alcune informazioni sui processi CLI attualmente esistenti.
WHY	spiega perché il comando precedente è fallito.

Comandi per il controllo delle sequenze

ECHO	visualizza il messaggio indicato come argomento.
EXECUTE	esegue un file di comandi.
FAILAT	blocca una sequenza di comandi se un programma restituisce un valore maggiore o uguale al numero specificato.
IF	verifica le condizioni indicate all'interno di una sequenza di comandi.
LAB	definisce un'etichetta (vedere SKIP).
QUIT	esce da una sequenza di comandi con un particolare codice d'errore.
SKIP	compie, all'interno di una sequenza, un salto in avanti fino a trovare un'etichetta (vedere LAB).
WAIT	attende che sia trascorso un determinato periodo di tempo o che venga raggiunta una particolare data.

Gestione del sistema e della memoria

ASSIGN	assegna un nome logico di device a una directory del filing system.
DATE	mostra o imposta la data e l'ora del sistema.
DISKCOPY	copia il contenuto di un intero floppy disk in un altro.
FAULT	visualizza il messaggio corrispondente al codice d'errore indicato.
FORMAT	formatta e inizializza un nuovo disco da 3,5".
INFO	fornisce le informazioni riguardanti il filing system.
INSTALL	fa sì che un disco formattato sia in grado di eseguire il bootstrap.

MOUNT	rende disponibile un nuovo device.
RELABEL	cambia il nome di volume a un disco.

Comandi per i programmatori

Sistema di sviluppo

ALINK	collega sezioni di codice in un file per l'esecuzione (vedere JOIN).
ASSEM	assembla il linguaggio del MC68000.
DOWNLOAD	trasferisce programmi nell'Amiga.
READ	legge informazioni dalla porta parallela o dalla linea seriale e le memorizza in un file.

Capitolo 3

ED - L'editor di schermo

Questo capitolo descrive come si usa l'editor di schermo ED. L'uso di questo programma serve a modificare o creare file di testo.

3.1 Introduzione a ED
3.2 Comandi immediati
3.2.1 Controllo del cursore
3.2.2 Inserimento di testo
3.2.3 Cancellazione di testo
3.2.4 Scrolling
3.2.5 Ripetizione immediata del comando
3.3 Comandi estesi
3.3.1 Controllo del programma
3.3.2 Controllo del blocco
3.3.3 Spostamento della posizione corrente del cursore
3.3.4 Ricerca e sostituzione
3.3.5 Modifica del testo
3.3.6 Ripetizione dei comandi estesi
3.4 Tavola di consultazione rapida

3.1 Introduzione a ED

Si può usare l'editor ED per creare un nuovo file o per modificarne uno già esistente. Il testo viene visualizzato sullo schermo e può scorrere sia orizzontalmente che verticalmente.

ED accetta il seguente template:

```
ED ''FROM/A,SIZE/K''
```

Per esempio si può attivare ED scrivendo

```
ED marco
```

ED tenta di aprire il file chiamato "marco", che è il file FROM, e, se ci riesce, lo carica in memoria visualizzandone sullo schermo le prime linee. Se il file non esiste, ED fornisce uno schermo vuoto pronto per accettare nuove informazioni. Per modificare il buffer riservato al testo nel quale ED mantiene il file, si deve indicare un valore opportuno dopo la keyword SIZE, come per esempio

```
ED marco SIZE 45000
```

La dimensione iniziale è basata sulla grandezza del file da modificare, con un minimo fissato in 40000 byte.

Nota: *non si può elaborare qualunque tipo di file con ED. Per esempio, i file contenenti un codice espresso in numeri binari non vengono accettati. Per accedere a file di questo tipo si deve ricorrere all'editor EDIT.*

ATTENZIONE: ED aggiunge sempre un LF (line feed) al termine del file.

Quando ED è in esecuzione, la linea inferiore dello schermo viene impiegata per i messaggi e per introdurre le linee di comando. I messaggi d'errore appaiono in tale area e vi rimangono fino a quando non viene impartito un altro comando ED.

I comandi ED sono divisi in due categorie:

- comandi immediati
- comandi estesi

I primi si usano in **modo immediato**; i secondi in **modo esteso**. Quando viene mandato in esecuzione, ED si configura in modo immediato. Per entrare nel modo esteso si deve far ricorso alla pressione del tasto ESC. In seguito, quando ED ha eseguito la linea di comandi, ritorna automaticamente in modo immediato.

In modo immediato, ED esegue subito i comandi. I comandi immediati vengono impartiti per mezzo di un singolo tasto o di una "combinazione control", che si realizza tenendo premuto il tasto CTRL contemporaneamente a una determinata lettera. Così CTRL-M significa premere CTRL assieme al tasto M.

Nel modo esteso, qualunque cosa si scriva appare nella linea di comando. ED esegue i comandi solo quando la linea è stata completata. Più comandi estesi possono essere scritti in una singola linea di comando e raggruppati in modo da essere ripetuti automaticamente. Infine la maggior parte dei comandi immediati ha un corrispondente comando esteso.

ED cerca di mantenere lo schermo aggiornato. Ciò nonostante, se viene impartito un altro comando mentre sta riscrivendo lo schermo, ED esegue prima il comando e poi aggiorna lo schermo quando esiste una disponibilità di tempo. La linea corrente viene sempre visualizzata per prima ed è quindi sempre aggiornata.

3.2 Comandi immediati

Questa sezione descrive il tipo di comandi che ED esegue immediatamente. I comandi immediati si occupano delle seguenti operazioni:

- controllo del cursore
- inserimento di testo
- cancellazione di testo
- scrolling del testo
- ripetizione dei comandi

3.2.1 Controllo del cursore

Per muovere il cursore in una qualunque direzione, si usano gli appositi tasti di controllo del cursore. Se il cursore si trova nella parte destra dello schermo, ED esegue uno scroll del testo verso sinistra per rendere visibile il resto del testo. L'editor esegue gli scroll verticali una linea alla volta e quelli orizzontali dieci caratteri per volta. Non si può muovere il cursore oltre l'inizio o la fine del file, oppure oltre il limite sinistro del testo.

CTRL-], cioè CTRL premuto assieme alla parentesi quadra chiusa "]", porta il cursore al limite destro della linea corrente. Se il cursore si trova già al limite destro, CTRL-] lo muove al limite sinistro, operando se necessario uno scroll. In modo simile agisce CTRL-E che porta il cursore all'inizio della prima linea dello schermo oppure, se già vi si trova, alla fine dell'ultima.

CTRL-T porta il cursore all'inizio della parola successiva. CTRL-R sposta il cursore allo spazio che segue la parola precedente. In entrambi i casi il testo, se necessario, viene spostato orizzontalmente o verticalmente.

Il tasto TAB muove il cursore alla successiva posizione di tabulazione, che è un multiplo della tabulazione iniziale (di default 3), ma non inserisce caratteri TAB nel file.

3.2.2 Inserimento di testo

Mentre è attivo il modo immediato, ED si trova anche in modo INSERIMENTO, e ogni carattere digitato viene inserito nella posizione del cursore. ED non possiede un modo di sovrascrittura. Per sostituire una parola

o una linea bisogna prima cancellarla e poi inserire il nuovo testo al suo posto. Qualunque lettera digitata in modo immediato appare nella **posizione corrente del cursore**, a meno che la linea non sia troppo lunga (una linea può contenere un massimo di 255 caratteri). Se si cerca di scrivere una linea più lunga del limite massimo, ED rifiuta i caratteri in eccesso e visualizza il messaggio

`Line too long` (Linea troppo lunga)

Comunque, operando con linee più corte, ED sposta qualunque carattere alla destra del cursore per far posto al nuovo testo. Se la linea eccede le dimensioni dello schermo, i caratteri più a sinistra scorrono "uscendo" dall'area visualizzata. Successivamente ED visualizza ancora la fine della linea operando uno scroll orizzontale. Se si muove il cursore oltre la fine della linea, per esempio con TAB o con i tasti di controllo del cursore, ED inserisce degli spazi tra la fine della linea e qualunque nuovo carattere venga inserito.

Per dividere la linea corrente dalla posizione del cursore, e generarne una nuova, si deve premere RETURN. Se il cursore si trova alla fine della linea, ED crea una nuova linea vuota dopo quella corrente. In alternativa, si può premere CTRL-A per generare una nuova linea vuota dopo quella corrente, senza che quest'ultima venga divisa. In ogni caso il cursore appare nella nuova linea alla posizione indicata come margine sinistro (inizialmente la prima colonna).

Per assicurarsi che ED generi automaticamente un CR (Carriage Return, ritorno carrello) a una certa posizione dello schermo, si può impostare un margine destro. Se questo è stato fatto, ogni volta che viene scritta una linea che supera questo margine, ED termina la linea prima dell'ultima parola, e sposta sia la parola sia il cursore alla linea successiva. Questa operazione viene chiamata "word wrap" (occorre notare che se è presente una linea senza spazi, ED non riesce a determinare dove tagliare la parola e il margine automatico non può lavorare nella maniera corretta). In particolare, se viene digitato un carattere mentre il cursore si trova al termine della linea e al margine destro, ED genera automaticamente una nuova linea. Se il carattere digitato è diverso da uno spazio, ED sposta la parola incompleta all'inizio della linea sottostante appena creata. Comunque, se viene inserita una parola quando il cursore NON si trova al termine della riga (cioè quando ha del testo alla sua destra), l'impostazione del margine destro non avrà effetto. Inizialmente il margine destro è posizionato alla colonna 79. Può essere disabilitato per mezzo del comando EX (per ulteriori dettagli sull'impostazione dei margini riferirsi alla sezione 3.3.1, "Controllo del programma").

Se il testo viene scritto in minuscolo invece che in maiuscolo, o viceversa, si può cambiare forma per mezzo di CTRL-F. Per fare questo è necessario portare il cursore sopra la lettera da correggere e premere CTRL-F. Se la lettera è minuscola diventerà maiuscola, se è maiuscola diventerà minusco-

la. Quando il cursore si trova sopra qualcosa che non è una lettera (per esempio uno spazio o un simbolo) CTRL-F non ha alcun effetto.

CTRL-F non cambia soltanto la forma delle lettere, ma muove anche il cursore di uno spazio verso destra, puntando in tal modo al carattere successivo, anche se non ha operato un cambiamento di forma. Se il carattere successivo è una lettera, si può premere nuovamente CTRL-F per farne cambiare la forma; il comando può essere ripetuto finché non sono state cambiate tutte le lettere della linea. Se si continua a premere CTRL-F anche dopo l'ultima lettera della riga, il cursore continua a spostarsi verso destra anche se non c'è più nulla da cambiare. Per esempio, se si sta scrivendo la linea

```
I Giovani e gli Anziani camminavano fianco a fianco
```

e si tiene premuto CTRL-F, la linea diventa

```
i gIOVANI E GLI aNZIANI CAMMINAVANO FIANCO A FIANCO
```

D'altra parte la linea seguente:

```
IF <file><= X
```

diventa

```
if <FILE><= x
```

dove le lettere passano da maiuscolo a minuscolo e viceversa, mentre i simboli rimangono inalterati.

3.2.3 Cancellazione di testo

Il tasto BACKSPACE cancella il carattere a sinistra del cursore, il quale viene poi spostato di una posizione a sinistra, a meno che non si trovi all'inizio della linea. Se è necessario, ED opera uno scroll. Il tasto DEL cancella il carattere che si trova nella posizione corrente del cursore lasciando inalterata la posizione del cursore. Come in ogni cancellazione, i caratteri che rimangono sulla linea vengono spostati indietro, e il testo che non era visibile oltre il limite destro dello schermo diventa visibile.

L'azione di CTRL-O dipende dal carattere sul quale si trova il cursore: se questo carattere è uno spazio, CTRL-O cancella tutti gli spazi della linea che precedono il carattere successivo. Altrimenti cancella tutti i caratteri successivi, facendo scorrere il testo verso sinistra, finché non trova uno spazio.

CTRL-Y cancella tutti i caratteri dal cursore fino al termine della linea.
CTRL-B cancella tutta la linea corrente. Per cancellare blocchi di testo si possono usare i comandi estesi.

3.2.4 Scrolling

Oltre allo scrolling verticale una linea alla volta ottenuto muovendo il cursore al limite dello schermo, si può operare uno scrolling verticale di 12 linee alla volta utilizzando i tasti CTRL-U e CTRL-D.
CTRL-D muove il cursore alle linee precedenti, e sposta il testo verso il basso; CTRL-U sposta il testo verso l'alto e muove il cursore alle linee successive del file.
CTRL-V visualizza il contenuto dell'intero schermo, nel caso che un programma diverso dall'editor l'abbia alterato. Comunque, nell'uso tipico, i messaggi derivanti da altri processi appaiono nella finestra dietro quella dell'editor.

3.2.5 Ripetizione immediata del comando

L'editor ricorda qualunque linea di comando sia stata digitata. Per ripetere in qualunque momento un insieme di comandi estesi, si può premere CTRL-G. Per esempio si può impostare un comando di ricerca come comando esteso. Se la prima stringa che appare non è quella cercata, basta premere CTRL-G per proseguire la ricerca. Si possono impostare ed eseguire complessi insiemi di comandi più volte.
Nota: *quando si indica un comando esteso come un gruppo di comandi con un contatore delle ripetizioni, ED ripete i comandi contenuti nel gruppo quel dato numero di volte, ogni volta che si preme CTRL-G. Per ulteriori informazioni sui comandi estesi, fate riferimento alla prossima sezione.*

3.3 Comandi estesi

Questa sezione descrive i comandi disponibili in modo esteso. I comandi realizzano le seguenti funzioni:

- controllo del programma
- controllo dei blocchi
- movimento
- ricerca di testo
- sostituzione di testo
- modifica del testo
- inserimento di testo

• cancellazione di testo
• scrolling del testo
• ripetizione dei comandi

Per usare i comandi estesi si deve premere il tasto ESC. L'input seguente appare nella linea di comando nella parte inferiore dello schermo. Si possono correggere gli errori con BACKSPACE nel modo normale. Per terminare una linea di comando si può premere sia ESC, sia RETURN. Se si preme ESC l'editor rimane in modo esteso dopo ever eseguito la linea di comando, mentre usando RETURN viene attivato il modo immediato. Per lasciare la linea di comando vuota, basta premere RETURN dopo ESC, in modo da tornare in modo immediato.

I comandi estesi sono composti da uno o due caratteri, sia in maiuscolo sia in minuscolo. Si possono impartire più comandi sulla stessa linea, avendo cura di separarli per mezzo di un punto e virgola (;). I comandi sono seguiti a volte da un argomento, come un numero o una stringa. Una stringa è una sequenza di lettere introdotta e terminata da un delimitatore, che può essere qualunque carattere eccetto le lettere, i numeri, lo spazio, il punto e virgola e i segni di maggiore e minore. Alcune stringhe valide potrebbero quindi essere:

```
/felice/
!23 metri!
:ciao:
''1/2''
```

La maggior parte dei comandi immediati dispone di una corrispondente versione estesa. Per averne una lista completa, fate riferimento alla tavola dei comandi estesi alla fine di questo capitolo.

3.3.1 Controllo del programma

Questa sezione fornisce una spiegazione dei seguenti comandi per il controllo del programma: X (eXit, esci salvando), Q (Quit, esci), SA (SAve, salva), U (Undo, torna alla situazione precedente), SH (SHow, mostra), ST (Set Tab, inserisci il tabulatore), SL e SR (Set Left e Set Right, margine sinistro e margine destro) e EX (EXtend, estendi il margine).

Per comunicare all'editor il termine delle operazioni si usa X. Una volta che si è usato il comando di uscita, ED scrive il testo mantenuto in memoria nell'output, o file destinazione, e termina. Se si ispeziona il contenuto del file vi si troveranno tutte le modifiche che sono state apportate.

ED genera anche un file di sicurezza (backup) temporaneo in :T/ED-BACKUP. Questo file rimane invariato finché non si esce un'altra volta da ED,

momento nel quale viene generato un nuovo file di sicurezza che sovrascrive il precedente.

Per uscire dall'editor senza apportare nessuna modifica al file residente su disco, si usa il comando Q. Così facendo, ED termina immediatamente senza scrivere nel buffer e tralasciando qualunque modifica sia stata effettuata. Per questo motivo nel caso che sia stato alterato il contenuto del file, viene chiesto di confermare se si desidera realmente uscire.

Un ulteriore comando permette di creare una copia "istantanea" del file senza uscire da ED: è il comando SA. SA (SAve) salva il testo nel file indicato o, in assenza di un'indicazione, in quello corrente. Per esempio,

```
SA !:doc/testo_salvato!
```

oppure

```
SA
```

Il comando SA è particolarmente utile in aree geografiche soggette a cadute di tensione o a sovratensioni transitorie.

Nota: *SA seguito da Q equivale al comando X.*

Se si opera una qualunque modifica tra i comandi SA e Q appare il seguente messaggio:

```
Edits will be lost-type Y to confirm:
```

(Quello che avete scritto andrà perso - digitate Y per confermare)

Se non sono state apportate modifiche, ED termina immediatamente lasciando invariato il contenuto del file sorgente. SA è utile anche perché permette di indicare un file diverso da quello corrente. Di conseguenza è possibile creare copie in fasi di lavoro diverse e porle in file o directory differenti.

Per eliminare gli effetti dell'ultima modifica riportando il testo nelle condizioni precedenti (Undo) si usa il comando U. L'editor crea una copia della linea nella quale si trova il cursore e modifica quella ogni volta che si aggiunge o si cancella un carattere. ED pone la copia con le modifiche nel file quando si muove il cursore oltre la linea corrente (sia con i tasti per il controllo del cursore sia cancellando o inserendo una linea). Inoltre ED sostituisce la copia quando opera uno scrolling sia verticale che orizzontale. Il comando U elimina la copia con le modifiche e usa in sua vece la vecchia versione della linea corrente.

ATTENZIONE: ED non recupera una linea che sia stata cancellata. Una volta che ci si sposta dalla linea corrente, il comando U non può ricomporre il testo originario.

Il comando SH viene usato per mostrare lo stato dell'editor. Lo schermo visualizza informazioni quali il valore dei punti di tabulazione, i margini attuali, le indicazioni di inizio e fine blocco e il nome del file che si sta attualmente usando.

I tab(ulatori) sono posti inizialmente ogni tre colonne. Per modificare l'impostazione corrente dei tab, si usa il comando ST (Set Tab) seguito da un numero "n", che indica ogni quante colonne ne deve essere posto uno.

Il margine sinistro e quello destro vengono impostati rispettivamente per mezzo dei comandi SL e SR (Set Left e Set Right), seguiti da un numero che indica la colonna a partire dalla quale hanno effetto. Il margine sinistro non deve trovarsi oltre la larghezza dello schermo.

Per estendere i margini si usa il comando EX (EXtend). Una volta che è stato impartito EX, ED non tiene conto del margine destro della linea corrente. Quando si sposta il cursore dalla linea corrente, ED attiva nuovamente i margini.

3.3.2 Controllo del blocco

Per spostare, inserire o cancellare una parte del testo si possono usare i comandi per il controllo del blocco, descritti in questa sezione.

Si può specificare un blocco di testo con i comandi BS (Block Start, inizio blocco) e BE (Block End, fine blocco). Per far questo è sufficiente muovere il cursore in un qualunque punto della prima linea che si desidera faccia parte del blocco, e quindi impartire il comando BS. Poi si deve spostare il cursore nell'ultima linea del blocco, usando i comandi per il controllo del cursore oppure un comando di ricerca, e infine impartire il comando BE che segna la fine del blocco.

Nota: *una volta che è stato definito un blocco con BS e BE, se si effettua una qualunque modifica al testo, l'inizio e la fine del blocco diventano nuovamente indefiniti. L'unica eccezione è rappresentata dall'uso di IB (Insert Block, inserisci il blocco).*

Per specificare una linea come blocco corrente, è sufficiente portare il cursore sulla linea voluta, premere ESC e digitare:

```
BS;BE
```

La linea corrente diventa anche il blocco corrente.

Nota: *non si può iniziare o terminare un blocco nel mezzo di una linea. Per farlo, occorre prima separarla premendo RETURN.*

Una volta che è stato indicato un blocco, se ne può ottenere una copia in un'altra parte del file con il comando IB (inserisci il blocco). Quando si impartisce il comando IB, ED inserisce una copia del blocco immediatamente

dopo la linea corrente. Si possono inserire più copie del blocco purché non si modifichi il testo o si cancelli il blocco.

Per cancellare un blocco si usa il comando DB (Delete Block, cancella il blocco). DB cancella il blocco di testo definito per mezzo dei comandi BS e BE. Inoltre, quando si cancella un blocco, i valori di inizio e fine blocco diventano indefiniti. Questo significa che non si può cancellare un blocco e poi inserirne una copia (DB seguito da IB), mentre si può inserire una copia di un blocco e poi cancellare il blocco stesso (IB seguito da DB).

Gli indicatori di inizio e fine blocco possono anche essere usati per ricordare una posizione all'interno di un file. Il comando SB (Show Block, mostra il blocco) riposiziona la finestra dello schermo in modo che la prima linea del blocco si trovi all'inizio dello schermo.

Per trasferire un blocco in un altro file si usa il comando WB (Write Block, scrivi il blocco). Questo comando accetta una stringa che rappresenta il nome del file. Per esempio:

```
WB !:doc/esempio!
```

scrive il contenuto del blocco nel file "esempio" della directory ":doc". Bisogna ricordare che, se si usa la barra (/) all'interno del filename per la suddivisione in directory e file, non è possibile impiegarla come delimitatore. ED crea un file con il nome che è stato indicato, distruggendo un eventuale file dotato dello stesso nome, e vi memorizza il blocco.

Il comando IF (Insert File, inserisci il file) permette di inserire un file all'interno di quello corrente. ED legge in memoria il contenuto del file il cui nome è stato dato come argomento a IF, e lo pone immediatamente dopo la linea corrente. Per esempio,

```
IF !:doc/esempio!
```

inserisce il file ":doc/esempio" in quello corrente, alla linea che segue immediatamente quella corrente.

3.3.3 Spostamento della posizione corrente del cursore

Il comando T (Top, sommità) sposta il cursore all'inizio del file, in modo che la prima linea del file sia la prima dello schermo. Il comando B (Bottom, fondo) sposta il cursore alla fine del file, in modo che l'ultima linea del file sia la linea più in alto dello schermo.

I comandi N (Next, prossima) e P (Previous, precedente) muovono il cursore all'inizio della linea che segue e di quella che precede. I comandi CL e CR (Cursor Left e Cursor Right) spostano il cursore di una posizione a sinistra e di una a destra, mentre CE (Cursor End) porta il cursore al termine

della linea corrente e CS (Cursor Start) all'inizio.

Il comando M (Move, sposta) invia il cursore a una determinata linea; basta scrivere M seguito dal numero della linea che deve diventare quella corrente. Per esempio,

```
M 503
```

muove il cursore alla cinquecentotreesima linea del file. Il comando M rappresenta un modo veloce per raggiungere un punto conosciuto del proprio file. È anche possibile spostarsi alla linea desiderata impartendo ripetutamente un comando N, ma è un metodo molto più lento.

3.3.4 Ricerca e sostituzione

In alternativa a quanto abbiamo visto in precedenza, è possibile spostarsi a una particolare posizione usando il comando F (Find, trova) seguito da una stringa che rappresenta il testo da ricercare. La ricerca inizia dal carattere successivo a quello coperto dal cursore e continua in avanti attraverso tutto il file. Se la stringa viene trovata, appare al suo inizio il cursore. La stringa che si vuole ricercare deve essere racchiusa tra virgolette o tra qualunque altro delimitatore ("/", ".", "!" e così via). Per combaciare, le stringhe devono essere della stessa forma (cioè il maiuscolo deve combaciare con il maiuscolo e il minuscolo con il minuscolo), a meno che non sia stato usato il comando UC (vedere più avanti).

Per operare una ricerca all'indietro attraverso il testo, si deve usare il comando BF (Backward Find, ricerca all'indietro) nello stesso modo di F. BF ricerca l'ultima posizione in cui si trova la stringa prima del cursore, cioè ricerca la stringa alla sinistra del cursore in tutte le linee precedenti, fino all'inizio del file. Per trovare la prima posizione si può usare T (inizio del file) seguito da F; per trovare l'ultima occorre usare B (fine del file) seguito da BF.

Il comando E (Exchange, scambio) accetta come argomenti due stringhe separate da un carattere delimitatore, e sostituisce la prima stringa con la seconda. Per esempio,

```
E /cane/gatto/
```

cerca la parola "cane" e la cambia in "gatto". L'editor inizia la ricerca della prima stringa dalla posizione corrente del cursore in poi, e prosegue attraverso il file. Dopo che la sostituzione è avvenuta il cursore si sposta alla fine della nuova stringa.

Si può indicare una stringa vuota digitando due delimitatori non separati da nulla. Se la prima stringa, o "stringa di ricerca", è vuota, l'editor inserisce la seconda stringa nella posizione corrente del cursore. Se è invece la seconda

stringa a essere vuota, la stringa ricercata viene scambiata con niente, cioè viene cancellata.

Nota: *ED ignora le impostazioni dei margini mentre effettua la ricerca e sostituzione nel testo.*

Il comando EQ (Exchange and Query, sostituzione con verifica) è una variante del comando E. Quando si usa EQ, ED chiede se si desidera che la sostituzione di stringhe abbia luogo. Questo si rivela molto utile quando la sostituzione deve avvenire in certe circostanze e non in altre. Per esempio, dopo avere digitato

```
EQ /cane/gatto/
```

appare nella linea di comando il seguente messaggio

```
Exchange?
```

che significa "Sostituisco?". Se si risponde con N il cursore si sposta dopo la stringa di ricerca; se invece si risponde con Y lo scambio avviene normalmente. Di solito si usa EQ solo se la stringa da cambiare si trova più di una volta.

Nel corso della ricerca, i comandi di ricerca e sostituzione operano una distinzione tra maiuscolo e minuscolo. Se si vuole che questa distinzione non avvenga più, si può usare il comando UC. Una volta che è stato impartito UC, la stringa di ricerca "gatto" combacia con "Gatto", "GATTO", "gaTTo" e così via. Usando il comando LC, ED ritorna a distinguere tra il maiuscolo e il minuscolo.

3.3.5 Modifica del testo

Il comando E non può essere usato per inserire una nuova linea nel testo; per far questo esistono i comandi I e A. Di seguito al comando I (Insert before, inserisci prima) deve essere posta la stringa che si desidera diventi una nuova linea; ED la inserirà prima della linea corrente. Per esempio,

```
I /Inserisci questa PRIMA della linea corrente/
```

inserisce la stringa "Inserisci questa PRIMA della linea corrente" come una nuova linea separata dal resto, prima della linea che contiene il cursore. Il comando A (insert After, inserisci dopo) si usa nello stesso modo eccetto per il fatto che ED inserisce la nuova linea dopo quella corrente. Cioè,

```
A /Inserisci questa DOPO la linea corrente/
```

inserisce la stringa "Inserisci questa DOPO la linea corrente" come una nuova linea dopo quella che contiene il cursore.

Per dividere la linea corrente nel punto dove si trova il cursore, si usa il comando S (Split, spezza). Il comando S in modo esteso equivale alla pressione del tasto RETURN in modo immediato (vedere la sezione 3.2.2 per ulteriori informazioni sulla divisione delle linee).

Il comando J (Join, unisci) congiunge la linea seguente con la fine di quella corrente.

Il comando D (Delete, cancella) cancella la linea corrente nello stesso modo in cui lo fa CTRL-B in modo immediato. Il comando DC cancella il carattere che precede il cursore nello stesso modo di DEL.

3.3.6 Ripetizione dei comandi estesi

Per ripetere qualunque comando un certo numero di volte è sufficiente scrivere prima del comando la cifra con il valore desiderato. Per esempio,

```
4 E /salta/brilla/
```

sostituisce "salta" con "brilla" nei primi quattro casi che si incontrano. ED verifica lo schermo dopo ogni comando. Il comando RP (RePeat, ripeti) permette di ripetere un comando finché ED non incontra un errore, come il raggiungimento della fine del file. Per esempio,

```
T; RP E /salta/brilla/
```

sostituisce "salta" con "brilla" ogni volta che lo incontra. Va notato che si è usato il comando T per partire dall'inizio del file ed essere sicuri che TUTTE le volte in cui si incontra la parola "salta" questa venga sostituita, altrimenti la sostituzione avverrebbe solo nei casi in cui la parola si trova dopo il cursore.

I gruppi di comandi possono essere eseguiti in modo ripetitivo se si ha l'accortezza di racchiuderli tra parentesi. Si possono anche nidificare gruppi di comandi all'interno di altri gruppi. Per esempio,

```
RP ( F /trelinee/; 3 A // )
```

inserisce tre linee vuote (copia la stringa nulla) dopo ogni linea contenente la stringa "trelinee". Questo comando agisce solamente dalla posizione del cursore fino al termine del file. Se si desidera che il comando operi per ogni linea bisogna prima spostare il cursore all'inizio del file.

Bisogna notare che alcune combinazioni sono possibili ma inutili. Per esempio,

```
RP SR 60
```

imposta il margine destro a 60 *ad infinitum*. Comunque, per interrompere una qualunque sequenza di comandi estesi, basta premere un tasto mentre il comando sta agendo. Se si verifica un errore, ED abbandona la sequenza di comandi.

3.4 Tavola di consultazione rapida

Tasti speciali

Comando	**Azione**
BACKSPACE	Cancella il carattere alla sinistra del cursore
DEL	Cancella il carattere sotto il cursore
ESC	Entra nel modo esteso
RETURN	Divide la linea dove c'è il cursore e ne crea una nuova
TAB	Muove il cursore alla successiva posizione di tabulazione (NON inserisce un carattere TAB)
<freccia-su>	Muove il cursore in su
<freccia-giù>	Muove il cursore in giù
<freccia-sinistra>	Muove il cursore a sinistra
<freccia-destra>	Muove il cursore a destra

Comandi immediati

Comando	**Azione**
CTRL-A	Inserisce una linea
CTRL-B	Cancella una linea
CTRL-D	Sposta il testo verso il basso
CTRL-E	Muove il cursore all'inizio o alla fine dello schermo
CTRL-F	Modifica dal maiuscolo al minuscolo e viceversa
CTRL-G	Ripete l'ultimo comando esteso
CTRL-H	Cancella un carattere alla sinistra del cursore (BACKSPACE)
CTRL-I	Muove il cursore verso destra, alla successiva posizione di tab
CTRL-M	RETURN
CTRL-O	Cancella le parole o gli spazi
CTRL-R	Sposta il cursore alla fine della parola precedente
CTRL-T	Sposta il cursore all'inizio della parola successiva
CTRL-U	Sposta il testo verso l'alto

CTRL-V	Verifica lo schermo
CTRL-Y	Cancella fino al termine della linea
CTRL-[	ESC (entra nel modo esteso)
CTRL-]	Cursore alla fine o all'inizio della linea

Comandi estesi

Questa è una lista completa dei comandi estesi, compresi quelli che sono solo versioni estese dei comandi immediati. Nella lista, /s/ indica una stringa, /s/t/ indica due stringhe che si scambiano e "n" indica un numero.

Comando	**Azione**
A /s/	Inserisce una linea dopo quella corrente
B	Muove il cusore alla fine del file
BE	Fine blocco dove si trova il cursore
BF /s/	Ricerca all'indietro
BS	Inizio blocco dove si trova il cursore
CE	Muove il cursore alla fine della linea
CL	Muove il cursore di una posizione verso sinistra
CR	Muove il cursore di una posizione verso destra
CS	Muove il cursore all'inizio della linea
D	Cancella la linea corrente
DB	Cancella il blocco
DC	Cancella il carattere sotto il cursore
E /s/t/	Cambia "s" in "t"
EQ /s/t/	Cambia ma prima chiede conferma
EX	Estende il margine destro
F /s/	Ricerca la stringa "s"
I /s/	Inserisce una linea prima di quella corrente
IB	Inserisce una copia del blocco
IF /s/	Inserisce il file "s"
J	Unisce la linea corrente con quella successiva
LC	Distingue nelle ricerche tra maiuscolo e minuscolo
M n	Muove il cursore alla linea numero "n"
N	Muove il cursore all'inizio della linea successiva
P	Muove il cursore all'inizio della linea precedente
Q	Esce dal programma senza salvare il testo
RP	Ripete finché non trova un errore
S	Divide la linea dove si trova il cursore
SA	Salva il testo nel file
SB	Mostra il blocco sullo schermo
SH	Fornisce informazioni

SL n	Imposta il margine sinistro
SR n	Imposta il margine destro
ST n	Imposta la distanza dei tab
T	Muove il cursore all'inizio del file
U	Rimette la linea corrente nelle condizioni precedenti alle modifiche
UC	Non distingue nelle ricerche tra maiuscolo e minuscolo
WB /s/	Scrive il blocco nel file "s"
X	Esce dal programma scrivendo il testo nel file

Capitolo 4

EDIT - L'editor di linea

Questo capitolo descrive in dettaglio come usare l'editor di linea EDIT. La prima parte introduce il lettore all'editor, la seconda fornisce una completa descrizione di EDIT. Alla fine del capitolo si trova una tavola di consultazione rapida dei comandi di EDIT.

4.1 Introduzione a EDIT
4.1.1 Attivazione di EDIT
4.1.2 Uso dei comandi di EDIT
4.1.2.1 La linea corrente
4.1.2.2 Numeri di linea
4.1.2.3 Selezione di una linea corrente
4.1.2.4 Qualificatori
4.1.2.5 Modifica dalla linea corrente
4.1.2.6 Cancellazione di linee intere
4.1.2.7 Inserimento di nuove linee
4.1.2.8 Ripetizione dei comandi
4.1.3 Abbandono di EDIT
4.1.4 Un esempio più complesso
4.2 Una descrizione completa di EDIT
4.2.1 Sintassi dei comandi
4.2.1.1 Nomi dei comandi
4.2.1.2 Argomenti
4.2.1.3 Stringhe
4.2.1.4 Stringhe multiple
4.2.1.5 Stringhe qualificate
4.2.1.6 Espressioni di ricerca
4.2.1.7 Numeri
4.2.1.8 Valori degli switch
4.2.1.9 Gruppi di comandi
4.2.1.10 Ripetizione di comandi
4.2.2 Uso di EDIT

4.2.2.1 Prompt
4.2.2.2 La linea corrente
4.2.2.3 Numeri di linea
4.2.2.4 Stringhe qualificate
4.2.2.5 Trattamento dell'output
4.2.2.6 Gestione della fine-del-file (EOF)
4.2.3 Raggruppamento funzionale dei comandi di EDIT
4.2.3.1 Selezione di una linea corrente
4.2.3.2 Inserimento e cancellazione di linee
4.2.4 Porzioni di linea
4.2.4.1 La porzione operativa
4.2.4.2 Operazioni sui singoli caratteri della linea corrente
4.2.5 Operazioni sulle stringhe nella linea corrente
4.2.5.1 Operazioni principali sulle stringhe
4.2.5.2 La stringa nulla
4.2.5.3 Variazioni con spostamento del puntatore
4.2.5.4 Cancellazioni di parti della linea corrente
4.2.6 Miscellanea di comandi per la linea corrente
4.2.6.1 Divisione e unione di linee
4.2.7 Ispezione di parte del sorgente: il comando Type
4.2.8 Controllo dei comandi e dei file di input e output
4.2.8.1 File di comandi
4.2.8.2 File di input
4.2.8.3 File di output
4.2.9 Iterazioni (Loop)
4.2.10 Operazioni globali
4.2.10.1 Impostazione delle modifiche globali
4.2.10.2 Disattivazione delle modifiche globali
4.2.10.3 Sospensione delle modifiche globali
4.2.11 Visualizzazione dello stato del programma
4.2.12 Termine dell'esecuzione di EDIT
4.2.13 Verifica della linea corrente
4.2.14 Miscellanea di comandi
4.2.15 Abbandono dell'editing interattivo
4.3 Tavola di consultazione rapida

4.1 Introduzione a EDIT

EDIT è un editor di testi che tratta file sequenziali una linea alla volta sotto il controllo di appositi comandi. EDIT scorre l'input – chiamato anche file sorgente – e trasferisce ogni linea, dopo che ha subito una qualunque modifica, a un file sequenziale di output: il file destinazione. L'esecuzione di

EDIT crea quindi una copia del file sorgente che contiene le modifiche richieste per mezzo dei comandi di editing.

Sebbene EDIT tratti il file sorgente sequenzialmente, partendo dall'inizio e procedendo verso la fine, possiede la capacità di spostarsi all'indietro di un limitato numero di linee. Questo è possibile perché EDIT non scrive immediatamente nel file destinazione le linee trattate ma le mantiene in una coda di output. La dimensione di questa coda dipende dalla quantità di memoria disponibile. Se si desidera mantenere in memoria un maggior numero di informazioni, si può selezionare un'opzione di EDIT (OPT) che serve per aumentare le dimensioni della "coda", e che descriveremo nella prossima sezione.

Si può scorrere il testo più di una volta.

I comandi di EDIT permettono di

a) cambiare parti del file sorgente
b) scrivere parti del file sorgente in altri file destinazione
c) inserire parti di testo provenienti da altri file sorgente.

4.1.1 Attivazione di EDIT

Questa sezione descrive il formato degli argomenti che possono essere specificati ogni volta che viene invocato EDIT. Il template di EDIT è il seguente:

```
FROM/A,TO,WITH/K,VER/K,OPT/K
```

Se non ci si ricorda il formato degli argomenti si può digitare:

```
EDIT ?
```

L'AmigaDOS visualizza il template completo sullo schermo (per ulteriori informazioni sull'uso dei comandi, vedere i capitoli 1 e 2 di questo manuale).

Adottando un altro metodo di descrizione, la sintassi del comando EDIT è quella che segue:

[FROM]<file>[[TO]<file>][WITH<file>][VER<file>][OPT Pn|Wn|PnWn]

L'argomento FROM rappresenta il file sorgente che si desidera elaborare. L'argomento deve apparire, ma la keyword relativa è opzionale; l'AmigaDOS accetta l'argomento FROM per posizione. Non viene quindi richiesta la presenza della keyword FROM.

L'argomento TO rappresenta il file destinazione, cioè il file nel quale EDIT

scrive il proprio output comprese le modifiche. Se viene omesso l'argomento TO, EDIT usa un file temporaneo a cui verrà imposto lo stesso nome del file FROM quando avremo concluso il lavoro. Se viene impartito il comando STOP, non avviene il cambiamento di nome del file temporaneo e il file FROM rimane invariato. Nella versione 1.2, EDIT, appena riceve il controllo, verifica se sul disco è presente la directory :T. Se non la trova, provvede a crearla automaticamente.

La keyword WITH rappresenta il file che contiene i comandi da impartire a EDIT. Se viene omesso l'argomento WITH, EDIT legge i propri comandi dal terminale.

La keyword VER rappresenta il file verso il quale EDIT manda i messaggi d'errore e le verifiche delle linee. Se l'argomento VER viene omesso, EDIT usa il terminale.

La keyword OPT permette di indicare le opzioni di EDIT. Le opzioni valide sono: P<n> che stabilisce il numero <n> di linee precedenti disponibili, e W<n> che fissa la lunghezza massima delle linee, dove <n> rappresenta il numero dei caratteri. A meno che non venga diversamente indicato l'AmigaDOS pone per default P40W120.

Si può usare OPT per aumentare o diminuire la dimensione della memoria disponibile. Per determinare la memoria disponibile EDIT usa P per W, cioè il numero delle linee precedenti moltiplicato per la lunghezza delle linee; quindi per cambiare la dimensione della memoria si possono regolare i numeri associati a P e W. P50 alloca più memoria del solito, mentre P30 ne alloca meno.

Ecco alcuni esempi d'attivazione di EDIT:

```
EDIT progl TO nuovo_progl WITH comandi_edit

EDIT progl OPT P50W240

EDIT progl VER file_ver
```

Nota: *al contrario di quanto avviene con ED, non si può usare EDIT per creare nuovi file. Se si tenta di creare un nuovo file, l'AmigaDOS restituisce un messaggio d'errore, poiché non riesce a trovare il file nella directory corrente.*

4.1.2 Uso dei comandi di EDIT

Questa sezione introduce alcuni dei comandi basilari di EDIT, omettendo però numerose funzioni avanzate. Un'esauriente descrizione della sintassi dei comandi e di tutte le funzioni disponibili è riportata nella sezione 4.2 "Una descrizione completa di EDIT".

4.1.2.1 La linea corrente

EDIT legge le linee del file sorgente e le scrive in quello destinazione; la linea che in un qualunque momento ha "in sospeso", cioè che ha letto ma che non ha ancora scritto, viene detta linea corrente. EDIT opera tutte le modifiche di testo nella linea corrente e inserisce prima di questa le linee nuove. Ogni volta che si inizia a operare con EDIT la linea corrente è la prima del file sorgente.

4.1.2.2 Numeri di linea

EDIT assegna a ogni linea presente nel file sorgente un unico numero di linea. Questo numero di linea non fa parte delle informazioni memorizzate nel file ma viene calcolato da EDIT contando le linee mentre vengono lette. Quando si usa EDIT ci si può riferire a una linea particolare indicando il suo numero di linea. Una linea che è stata letta mantiene il numero di linea originario per tutto il tempo che rimane nella memoria centrale, anche se sono state cancellate delle linee prima o dopo di essa o se ne sono state inserite di nuove. I numeri di linea rimangono invariati finché il file non viene letto nuovamente dal principio o finché non si rinumerano le linee usando il comando "=". EDIT assegna i numeri di linea ogni volta che inizia ad accedere a un file. È possibile dunque che i numeri di linea non siano gli stessi quando si accede nuovamente al file.

4.1.2.3 Selezione di una linea corrente

Per selezionare una linea corrente è possibile usare uno dei tre metodi seguenti:

a) per conteggio di linee
b) indicando il numero di linea
c) indicando il contesto

Questi tre metodi sono descritti di seguito.

Per conteggio di linee

I comandi N e P permettono di muoversi alle linee seguenti e precedenti. Per muoversi di un certo numero di linee in avanti oppure all'indietro bisogna indicare tale numero prima dei comandi N o P. Per muoversi alla linea seguente si può digitare:

```
N
```

Qualunque comando di EDIT può essere dato sia in maiuscolo che in minuscolo.

Per muoversi di quattro linee in avanti si può scrivere:

```
4N
```

in questo modo ci si sposta alla quarta linea dopo quella corrente.
Per muoversi alla linea precedente si può digitare:

```
P
```

Anche il comando P può essere preceduto da un numero. Per esempio, scrivendo

```
4P
```

si sposta la linea corrente alla quarta linea che la precede. È possibile spostarsi soltanto a linee precedenti che EDIT non abbia ancora scritto nel file di output. Normalmente EDIT permette di tornare indietro fino a 40 linee di testo, ma si può modificare tale limite, usando l'opzione P quando si manda in esecuzione il programma (per ulteriori informazioni sull'opzione P, vedere la sezione 4.1.1 in questo stesso capitolo).

Spostamento a un particolare numero di linea
Il comando M (Move, sposta) permette di selezionare una nuova linea corrente indicandone il numero. Si deve scrivere M seguito dal numero di linea desiderato; per esempio il comando M45 comunica a EDIT di spostarsi alla linea 45. Se ci si trova oltre la linea 45, il comando torna indietro solo se la linea in questione si trova ancora nella memoria centrale.
Si possono combinare i due tipi di comandi visti in precedenza. Per esempio

```
M12; 3N
```

Per separare più comandi consecutivi in una stessa linea si deve interporre tra loro un punto e virgola (;).

Indicazione del contesto
Il comando F (Find, trova) è usato per individuare e selezionare una linea a seconda del suo contenuto. Per esempio,

```
F/Qualcosa/
```

fa sì che il programma ricerchi la linea che contiene "Qualcosa". La ricerca inizia dalla linea corrente e prosegue in avanti attraverso il file sorgente, finché non viene trovata la parola "Qualcosa". Se EDIT raggiunge la fine del sorgente senza averla trovata, viene visualizzato il seguente messaggio:

```
INPUT EXHAUSTED
```
(Sorgente terminato)

È possibile operare una ricerca all'indietro usando il comando BF Backward Find (ricerca all'indietro). Ecco un esempio:

```
BF/gira e rigira/
```

La ricerca ha inizio dalla linea corrente e prosegue all'indietro finché non viene trovata la linea desiderata. Se EDIT raggiunge l'inizio della coda di output senza trovare nulla, visualizza il seguente messaggio:

```
NO MORE PREVIOUS LINES
```
(non ci sono altre linee precedenti)

In questi esempi, il testo desiderato ("Qualcosa" e "gira e rigira") è racchiuso tra barre singole (/). Questo testo è chiamato stringa di caratteri. I caratteri che si usano per indicare l'inizio e la fine della stringa sono chiamati delimitatori. Nei casi affrontati sopra abbiamo usato come delimitatore "/". Possono essere usati come delimitatori un certo numero di caratteri speciali come ":" "." "," e "*"; naturalmente la stringa stessa non deve contenere il carattere delimitatore. EDIT ignora gli spazi tra il nome del comando e il primo delimitatore, ma considera spazi significativi quelli all'interno della stringa. Per esempio,

```
F /Ahi ahi ahi/
```

non considera né "ahiahi ahi", né "ahi ahiahi".

Se si usa un comando F senza argomenti, EDIT ripete la ricerca precedente. Per esempio,

```
F/passero/; N; F
```

cerca la seconda ricorrenza di "passero" lungo le linee.
Tra i due comandi F, è necessario interporre N, perché F comincia la ricerca partendo dalla linea corrente e di conseguenza, omettendo N, il secondo F avrebbe trovato la stessa linea del primo.

4.1.2.4 Qualificatori

La forma normale del comando F, descritta in precedenza, permette di cercare una linea che contenga la stringa data in qualunque suo punto. Per restringere la ricerca all'inizio o alla fine delle linee si può porre la lettera B o E (Begin, comincia, e End, finisci) prima della stringa. In questo caso si devono inserire uno o più spazi dopo la F. Per esempio,

```
F B/corvi neri/
```

indica che si vuol cercare la linea che inizia con "corvi neri", mentre

```
F E/orologio/
```

indica che si vuol cercare la linea che termina con "orologio". L'uso di B ed E velocizza la ricerca, e lo stesso accade quando si pongono altre condizioni alla ricerca, perché in questo modo EDIT deve considerare solamente una parte di ogni linea.

B ed E, così come sono stati usati in precedenza, sono esempi di **qualificatori**, mentre l'intero argomento viene denominato **stringa qualificata**. Sono disponibili un certo numero di altri qualificatori; per esempio,

```
F P/seduti su un marciapiede/
```

indica che si vuol cercare la prossima linea che contiene esattamente le parole "seduti su un marciapiede". La linea richiesta non deve contenere altri caratteri, neanche prima o dopo la stringa data. Quando viene impartito questo comando, EDIT ricerca la prossima linea che contiene:

```
seduti su un marciapiede
```

mentre non prende in considerazione la linea:

```
seduti su un marciapiede.
```

Per trovare una linea vuota (cioè che non contiene assolutamente niente) è possibile usare una stringa vuota con il qualificatore P, per esempio

```
F P//
```

Si possono fornire più qualificatori in qualunque ordine.

4.1.2.5 Modifica della linea corrente

Questa sezione spiega come usare i comandi E, A e B per modificare il testo della linea corrente.

Sostituzione di stringhe

Il comando E sostituisce una stringa di caratteri della linea con un'altra. Per esempio:

```
E/Paese/Citta'/
```

rimuove la stringa "Paese" dalla linea corrente e la sostituisce con "Città".

Bisogna notare che si usa un solo carattere delimitatore per separare le due stringhe. Per cancellare una parte della linea (cioè sostituire il testo con niente) si può usare una seconda stringa nulla, come segue:

```
E/mostruoso//
```

I comandi A e B permettono di aggiungere nuovo testo alla linea corrente. Il comando A (After, dopo) inserisce una stringa dopo quella data. In modo simile, il comando B (Before, prima) inserisce una stringa prima di quella data. Per esempio, se la linea corrente contiene

```
Sette spose per sette fratelli
```

allora la sequenza di comandi seguente:

```
A/sette/cento/; B L/sette/trenta/
```

la modificherà in:

```
Settecento spose per trentasette fratelli
```

Se il qualificatore L fosse stato omesso dal precedente comando B, si sarebbe invece ottenuto:

```
Trentasettecento spose per sette fratelli
```

e questo perché la ricerca di una stringa procede di solito da sinistra verso destra e EDIT usa la prima ricorrenza che trova. Il qualificatore L è usato per specificare che la ricerca deve procedere da destra verso sinistra.

Se la prima stringa di un comando A, B o E è vuota, EDIT inserisce la seconda stringa all'inizio della linea, oppure alla fine se viene usato il qualificatore L.

Se si impartisce un comando A, B o E in una linea che non contiene la stringa fornita come primo argomento, appare sullo schermo, o nel file di verifica che è stato indicato all'inizio delle operazioni con EDIT, il seguente messaggio:

```
NO MATCH
```

a indicare che la ricerca della stringa si è svolta senza successo. Per i particolari che riguardano il file di verifica, riferirsi alla sezione 4.1.1, "Attivazione di EDIT".

4.1:2.6 Cancellazione di linee intere

Questa sezione descrive come rimuovere linee di testo dal proprio file. Per cancellare una serie di linee contigue è sufficiente indicarne i numeri in un comando D. Si dovrà digitare D seguito dal numero di linea. Aggiungendo uno spazio e un secondo numero dopo D, EDIT rimuove tutte le linee comprese tra il primo numero di linea e l'ultimo. Per esempio,

```
D97 104
```

cancella le linee da 97 a 104 comprese, lasciando la linea 105 come nuova linea corrente. Se non è seguito da un numero, D cancella la linea corrente. Per esempio,

```
F/torta/; D
```

cancella la linea che contiene "torta", mentre la linea successiva diventa la nuova linea corrente. Una ricerca qualificata e un comando di cancellazione possono essere usati in modo consecutivo, come si vede nell'esempio che segue:

```
F B/Il/; 4D
```

Questa sequenza di comandi cancella quattro linee, la prima delle quali è la linea che inizia con "Il".

Si possono anche usare un punto (.) o un asterisco (*) al posto dei numeri di linea. Il punto si riferisce alla linea corrente, mentre l'asterisco indica la fine del file. Per esempio,

```
D. *
```

cancella il resto del file sorgente, inclusa la linea corrente.

4.1.2.7 Inserimento di nuove linee

Questa sezione descrive come inserire testo in un file usando EDIT. Per inserire una o più linee di testo PRIMA della linea corrente si usa il comando I (Insert, inserisci). Si può impartire il comando I da solo, con un numero di linea, un punto (.) o un asterisco (*). EDIT inserisce il testo prima della linea corrente se si usa I da solo o seguito dal punto. Se si aggiunge un asterisco dopo I, il testo viene inserito alla fine del file (cioè prima della linea di fine file). Qualunque testo venga digitato, viene inserito prima della linea specificata.

Per indicare la fine dell'inserimento bisogna premere RETURN, scrivere Z

e premere un'altra volta RETURN. Per esempio,

```
I 468
I piccoli pesci del mare
mi portarono una risposta.
Z
```

inserisce le due linee di testo prima della linea 468.

Se si omette il numero di linea, EDIT inserisce il nuovo testo prima della linea corrente. Per esempio,

```
F/cucciolo/; I
Egli disse, ''Andro' e lo sveglierò', se...''
Z
```

Questo comando multiplo ricerca la linea che contiene "cucciolo" (che quindi diventa la nuova linea corrente) e inserisce la nuova linea specificata.

Dopo un comando I che specifica un numero di linea, la linea corrente diventa quella che ha quel dato numero; altrimenti la linea corrente rimane inalterata.

Per inserire un testo alla fine del file bisogna usare I*.

Per facilitare la digitazione, EDIT dispone del comando R (Replace, sostituisci) che è l'esatto equivalente di DI (D per Delete, cancella seguito da I per Insert, inserisci). Per esempio,

```
R19 26
In inverno quando i campi sono bianchi
Z
```

cancella le linee da 19 a 26 comprese poi inserisce il nuovo testo prima della linea 27, che diventa la nuova linea corrente.

4.1.2.8 Ripetizione dei comandi

Si possono usare i valori individuali di ripetizione, come negli esempi per N e D, con numerosi altri comandi EDIT. Inoltre si possono ripetere serie di comandi raggruppandoli tra parentesi come segue:

```
6(F P//; D)
```

cancella le prossime 6 linee vuote nel file sorgente. I gruppi di comandi non possono essere più lunghi di una linea di input.

4.1.3 Abbandono di EDIT

Per terminare una sessione di EDIT bisogna usare il comando W (Windup, chiudi). EDIT "vola" fino al termine del file sorgente, copiandolo in quello destinazione, e termina la sua esecuzione. A meno che non sia stato indicato

un file TO, EDIT cambia il nome del file temporaneo di output in quello del file FROM.

EDIT può accettare i comandi da diverse fonti. Nel caso più semplice accetta i comandi direttamente dal terminale (cioè dalla tastiera); questo è chiamato **livello di comando primario**. EDIT può, comunque, accettare i comandi da altre fonti, come i **file di comandi**, chiamati anche file WITH.

Si possono richiamare i file di comandi dall'interno di EDIT e ulteriori file di comandi dall'interno di questi stessi file per mezzo del comando C, in modo che ogni file nidificato diventi un livello di comando separato. EDIT ferma l'esecuzione di un file di comandi quando ne raggiunge la fine o quando incontra un un comando Q, restituendo poi il controllo al livello di comando precedente. Quindi, se EDIT incontra un comando Q in un file nidificato, torna a eseguire i comandi contenuti nel file del livello precedente. Se si digita Q al livello primario di comando, o se EDIT incontra Q in un file WITH, il programma termina l'esecuzione ed esce come se si fosse usato W.

Il comando STOP ferma l'esecuzione di EDIT senza ulteriori elaborazioni. In particolare, non viene scritta nessuna linea di output ancora presente in memoria e quindi il file destinazione risulta incompleto. Se si è indicato solamente l'argomento FROM, EDIT non sovrascrive il file sorgente con il file (incompleto) modificato. STOP viene usato solamente se si desidera che le correzioni non vengano effettuate.

EDIT scrive un file di sicurezza temporaneo in :T/ED-BACKUP quando si pone termine alle operazioni con W o Q. Questo file di backup rimane valido finché non si esce nuovamente da EDIT con uno di questi comandi, visto che verrebbe sovrascritto dal nuovo file di backup. Usando STOP il file di backup non viene creato.

4.1.4 Un esempio più complesso

Le più semplici necessità di editing possono essere già soddisfatte usando i comandi descritti finora. Questa sezione presenta un esempio che usa diversi comandi. Il testo tra parentesi che nell'esempio segue i comandi di editing è un commento, e non deve essere scritto; infatti EDIT non permette commenti nella linea di comando.

Supponiamo di disporre del seguente testo sorgente (con i numeri di linea):

```
1    Tweedledee and Tweedledum
2    agreed to a battle,
3    For Tweedledum said Tweedledee
4    ad spoiled his nice new rattle.
5
6    As black as a tar barrel
7    Which frightened both the heroes so
8    They quite forgot their quorell
```

e di eseguire i seguenti comandi EDIT:

`M1; E/dum/dee/; E/dee/dum/`	(l'ordine dei comandi E è importante!)
`N; E/a/A/; B/a /have /`	(ora alla linea 2)
`F B/ad/; B//H/`	(H all'inizio della linea)
`F P//; N; I`	(prima della linea che segue quella bianca)
`Just then flew down a monstrous crow,`	
`Z`	
`M6; 2(A L//,/; N)`	(virgole alla fine delle linee)
`F/quore/; E/quorell/quarrel./`	(F in effetti è inutile)
`W`	(fine)

Quello che risulta è il testo seguente, con i nuovi numeri di linea.

```
1    Tweedledum and Tweedledee
2    Agreed to have a battle,
3    For Tweedledum said Tweedledee
4    Had spoiled his nice new rattle.
5
6    Just then flew down a monstrous crow,
7    As black as a tar barrel,
8    Which frightened both the heroes so,
9    They quite forgot their quarrel.
```

Nota: *se si prova a elaborare questo file sorgente, si scopre che non è necessario usare i comandi dell'esempio precedente. Per esempio, nella seconda linea, è possibile usare il seguente comando:*

```
E/a/have a/
```

per produrre lo stesso risultato. In altre parole E sostituisce "a" con "have a", mentre B pone "have " prima di "a" per produrre "have a".

4.2 Una descrizione completa di EDIT

Dopo aver letto la prima parte di questo capitolo dedicato alle principali caratteristiche di EDIT, dovreste essere in grado di usare l'editor per semplici operazioni. Il resto di questo capitolo è costituito da una sezione di consultazione che fornisce una descrizione completa di tutte le peculiarità di EDIT. Può essere necessario consultare questa sezione se si ha qualche problema mentre si modifica un testo oppure se si desidera usare EDIT in una maniera più sofisticata.

Gli argomenti descritti in questa sezione sono i seguenti:

- Sintassi dei comandi
- Uso di EDIT
- Raggruppamenti funzionali dei comandi di EDIT
- Porzioni di linea
- Operazioni di stringa sulla linea corrente
- Miscellanea di comandi per la linea corrente
- Ispezione di parte del file sorgente: il comando Type
- Controllo dei comandi, file di input e di output
- Iterazioni
- Operazioni globali
- Visualizzazione dello stato del programma
- Termine dell'esecuzione di EDIT
- Verifica della linea corrente
- Miscellanea di comandi
- Abbandono dell'editing interattivo

4.2.1 Sintassi dei comandi

Un comando di EDIT è costituito dal nome del comando seguito da zero o più argomenti. Uno o più spazi possono apparire opzionalmente tra il nome di un comando e il primo argomento, tra argomenti che non siano stringhe e tra i vari comandi. Uno spazio è necessario solamente quando deve separare voci successive che altrimenti sarebbero trattate come una sola (per esempio due numeri).

EDIT riconosce la fine di un comando in uno dei seguenti modi: quando si preme RETURN, quando raggiunge la fine degli argomenti del comando, quando legge un punto e virgola (;) o una parentesi chiusa ()).

Le parentesi sono usate per delimitare i gruppi di comandi.

Il punto e virgola è usato per separare i comandi che appaiono sulla stessa linea. Questo è strettamente necessario solo in caso di ambiguità, quando un comando può avere un numero variabile di argomenti; infatti EDIT cerca di leggere sempre il comando più lungo possibile.

Le lettere maiuscole e minuscole, eccetto quando appaiono come parte di una stringa di caratteri, sono considerate da EDIT equivalenti.

4.2.1.1 Nomi dei comandi

Il nome di un comando è una sequenza di lettere o un singolo carattere speciale (per esempio #). Il nome alfabetico di un comando termina con qualunque carattere che non sia una lettera; inoltre solo i primi quattro caratteri sono significativi. Tra il nome del comando e i suoi argomenti possono apparire uno o più spazi; EDIT richiede la presenza di almeno uno

spazio quando un argomento di un nome che contiene lettere inizia con una lettera.

4.2.1.2 Argomenti

Le sezioni seguenti descrivono i sei differenti tipi di argomenti che si possono usare con i comandi EDIT:

- stringhe
- stringhe qualificate
- espressioni di ricerca
- numeri
- valori degli switch
- gruppi di comandi

4.2.1.3 Stringhe

Una stringa è una sequenza con un massimo di 80 caratteri racchiusa tra delimitatori. È possibile anche usare una stringa vuota (nulla). Una stringa nulla è esattamente ciò che il suo nome suggerisce: una stringa priva di caratteri, cioè due delimitatori che non racchiudono nulla (per esempio //). Il carattere che si sceglie come delimitatore non deve apparire all'interno della stringa stessa. Il delimitatore finale può essere omesso se è immediatamente seguito dalla fine della linea di comando.

Possono essere usati come delimitatori i caratteri seguenti:

/ . + - , ? : *

cioè i normali caratteri di interpunzione (eccetto ;) e i quattro operatori aritmetici.

Questi sono alcuni esempi di stringhe:

```
/A/
*Lago di Garda*
??
+Stringa senza delimitatore finale
```

4.2.1.4 Stringhe multiple

I comandi che usano due stringhe come argomenti, richiedono lo stesso delimitatore e non lo raddoppiano tra gli argomenti stessi. Un esempio è il comando A:

```
A /Re/Il Rosso /
```

In tutti questi tipi di comandi, la seconda stringa indica il testo da inserire. Se omettete la seconda stringa, EDIT la interpreta come stringa nulla. Se lo fate con i comandi A e B non accade niente, perché avete chiesto a EDIT di non aggiungere niente prima o dopo la prima stringa. Se invece omettete la seconda stringa dopo un comando E, EDIT cancella la prima stringa.

4.1.2.5 Stringhe qualificate

I comandi che svolgono ricerche nel testo, sia nella linea corrente sia scandendo tutto il file sorgente, accettano le stringhe qualificate. Una stringa qualificata è una stringa preceduta da zero o più qualificatori. I qualificatori sono lettere singole e possono apparire in qualunque ordine. Per esempio,

```
BU/Abc/
```

Tra i qualificatori non possono apparire spazi. Si può terminare una lista di qualificatori con un qualunque carattere delimitatore. I qualificatori disponibili sono B (Begin, inizio), E (End, termine), L (Left o Last, sinistro o ultimo), P (Precisely, precisamente) e U (Uppercase, maiuscolo).

4.2.1.6 Espressioni di ricerca

I comandi che ricercano una linea particolare nel file sorgente accettano come argomento una espressione di ricerca. Un'espressione di ricerca è una singola stringa qualificata. Per esempio,

```
F B/Ciao/
```

comunica a EDIT di cercare una linea che cominci con la stringa "Ciao".

4.2.1.7 Numeri

Un numero è una sequenza di cifre decimali. I numeri di linea sono un caso speciale di numeri e devono essere sempre maggiori di zero. Ovunque può apparire un numero di linea, si possono usare anche i caratteri "." e "*". Un punto rappresenta la linea corrente, mentre un asterisco rappresenta l'ultima linea del file sorgente. Per esempio,

```
M*
```

comunica a EDIT di muoversi alla fine del file sorgente.

4.2.1.8 Valori degli switch

I comandi che alterano gli switch di EDIT accettano un singolo carattere come argomento. Per esempio, in

```
V-
```

il segno meno (-) indica che EDIT deve sospendere la verifica. Se in seguito si digita V+, EDIT riprende nuovamente la verifica. Quindi si può considerare il segno + come "attiva" e quello - come "disattiva".

4.2.1.9 Gruppi di comandi

Per raggruppare una sequenza di comandi EDIT individuali occorre

racchiuderli tra parentesi. Per esempio la linea seguente:

```
(f/Tricheco/;e/Tricheco/Grosso mammifero marino/)
```

cerca la prima volta in cui si trova la parola "Tricheco" e la cambia in "Grosso mammifero marino". I gruppi di comandi, comunque, non possono essere più lunghi di una linea di input. Se si introduce un gruppo di comandi più lungo di una linea, EDIT accetta solamente i comandi fino alla fine della prima linea. Poi, visto che EDIT non troverà la parentesi chiusa alla fine della linea, appare il seguente messaggio di errore:

```
Unmatched parentesis
```

(Mancano le parentesi di chiusura)

Bisogna notare che è necessario usare le parentesi quando si intende ripetere un gruppo di comandi più di una volta.

4.2.1.10 Ripetizione dei comandi

EDIT accetta numerosi comandi preceduti da un numero decimale senza segno che ne indica la ripetizione. Per esempio,

```
24N
```

ripete 24 volte il comando N

Se viene indicato il valore 0, EDIT esegue il comando all'infinito o finché non raggiunge la fine del file. Per esempio, se si digita

```
0(e /are/ere/;n)
```

viene scambiato ogni "are" con "ere", fino al termine del file.

Si può specificare un valore di ripetizione per un gruppo di comandi nello stesso modo usato per i comandi singoli:

```
12(F/dalla/; E/dalla/dal/; 3N)
```

4.2.2 Uso di EDIT

Questa sezione descrive quello che succede quando si esegue EDIT. Spiega in particolare da dove proviene l'input e dove viene diretto l'output, cosa appare sullo schermo e cosa eventualmente si verifica nel proprio file dopo che EDIT è stato eseguito.

4.2.2.1 Prompt

Quando EDIT viene eseguito in modo interattivo – cioè con il file di comandi connesso alla tastiera e il file di verifica connesso a una finestra –

fa apparire un prompt quando è pronto per leggere una nuova linea di comandi. Se l'ultimo comando della linea precedente ha causato la generazione di output di verifica, EDIT non mostra il prompt.

Se si abilita la verifica accendendo lo switch V, EDIT verifica la linea corrente invece del prompt nelle seguenti circostanze:

- se non ha ancora verificato la linea corrente,
- se sono state apportate delle modifiche alla linea dall'ultima verifica,
- se è stata cambiata la posizione della finestra operativa.

Altrimenti, quando EDIT non verifica la linea corrente, visualizza i due punti (:) per indicare che è pronto per una nuova linea di comandi. I due punti sono il prompt abituale di EDIT.

EDIT non fornisce mai il prompt quando si stanno inserendo delle linee.

4.2.2.2 La linea corrente

EDIT legge le linee del file sorgente e le scrive in quello destinazione; la linea che in un qualunque momento ha "in sospeso", cioè che ha letto ma che non ha ancora scritto, viene detta linea corrente. Ogni comando impartito si riferisce alla linea corrente. EDIT inserisce le linee nuove prima della linea corrente. Ogni volta che si inizia a operare con EDIT la linea corrente è la prima del file sorgente.

4.2.2.3 Numeri di linea

EDIT identifica ogni linea presente nel file sorgente con un numero di linea unico. Questo numero di linea non fa parte delle informazioni memorizzate nel file ma viene calcolato da EDIT contando le linee mentre vengono lette. EDIT non assegna i numeri di linea alle nuove linee che l'utente inserisce nel file sorgente.

EDIT opera una distinzione tra linee originali e non. Le linee originali sono quelle che non sono state divise o inserite. I comandi che accettano un numero di linea come argomento si possono riferire solamente a linee originali. EDIT si muove in avanti o all'indietro fino a un certo limite, a seconda che il numero di linea digitato sia maggiore o minore del numero di linea corrente. EDIT passa sopra o cancella (se questo compito gli viene assegnato) le linee non originali mentre è alla ricerca di una certa linea originale.

Se si usa un punto (.) invece di un numero di linea, EDIT usa sempre la linea corrente, che si tratti di una linea originale o no (per un esempio, riferirsi alla sezione 4.1.2.6, "Cancellazione di linee intere").

Si possono rinumerare le linee per mezzo del comando "=". Questo assicura che tutte le linee dopo quella corrente siano originali. Digitate:

```
=15
```

per numerare la linea corrente come 15, la linea successiva come 16, la successiva ancora 17 e così via fino al termine del file, assegnando i numeri di linea anche a quelle non originali. Se non si aggiunge un numero al comando =, EDIT visualizza il messaggio:

```
Number expected after =
```
(È necessario un numero dopo l'=)

4.2.2.4 Stringhe qualificate

Per indicare il contesto in cui EDIT opera le ricerche, si possono usare le stringhe qualificate. EDIT accetta la stringa nulla che corrisponde al punto di ricerca iniziale, cioè l'inizio della linea, eccetto per i casi riportati di seguito. In assenza di un qualificatore, EDIT può trovare la stringa indicata come parametro in qualunque punto di una linea. I qualificatori dichiarano condizioni aggiuntive per rendere la ricerca più specifica. EDIT riconosce cinque qualificatori: B, E, L, P e U.

B

La stringa deve trovarsi all'inizio della linea. Questo qualificatore non può apparire con E, L o P.

E

La stringa deve trovarsi alla fine della linea. Questo qualificatore non può apparire con B, L o P. Se E viene usato con la stringa nulla, individua la fine della linea (cioè cerca un carattere nullo alla fine della linea).

L

La ricerca della stringa si svolge verso sinistra partendo dalla fine della linea invece che verso destra dall'inizio della linea. Se la stringa si trova nella linea più di una volta, questo qualificatore assicura che sia riconosciuta l'ultima ricorrenza invece della prima. L non può apparire con B, E o P. Se L viene usato con la stringa nulla, individua la fine della linea (cioè cerca un carattere nullo iniziando dalla fine della linea verso sinistra).

P

La linea deve contenere esattamente la stringa e nessun altro carattere. P non può apparire con B, E o L. Se P viene usato con la stringa nulla, viene individuata la prima linea vuota.

U

La stringa può apparire in una qualunque combinazione di maiuscole e minuscole, come se si trasformassero sia la stringa sia la linea in lettere maiuscole prima di confrontarle. Per esempio, indicando U con la stringa seguente,

```
/TRIcheco/
```

si può individuare una linea contenente

```
trichECO
```

così come qualunque altra combinazione di maiuscole e minuscole.

4.2.2.5 Trattamento dell'output

EDIT non scrive le linee modificate immediatamente nel file destinazione, ma le aggiunge a una coda di output in memoria centrale. Quando la memoria disponibile per queste linee si è esaurita, trasferisce nel file destinazione le linee poste all'inizio della coda secondo le necessità. Finché una linea è ancora nella coda in memoria centrale, è possibile indicarla per renderla nuovamente la linea corrente.

Si possono anche scrivere porzioni dell'output in un file destinazione diverso da quello TO. Quando si seleziona un file destinazione alternativo, EDIT vi scrive la coda delle linee per il file di output corrente.

4.2.2.6 Gestione della fine-del-file (EOF)

Quando EDIT raggiunge la fine del file, una linea end-of-file fittizia diventa quella corrente. Questa linea speciale ha un numero di linea pari al numero di linee del file più uno. EDIT verifica la linea mostrando il numero di linea e un asterisco.

Quando la linea di fine file è quella corrente, i comandi per apportare modifiche alla linea corrente e quelli per muoversi in avanti, producono un errore. Nonostante ciò, se comandi di questo tipo sono contenuti in un gruppo che deve essere ripetuto all'infinito, EDIT non fornisce alcun errore al raggiungimento della linea di end-of-file. Il comando E (scambio) è un esempio di comando che modifica la linea corrente, mentre N (linea successiva) è un esempio di comando per spostarsi in avanti.

4.2.3 Raggruppamento funzionale dei comandi di EDIT

Questa sezione contiene la descrizione di tutti i comandi EDIT suddivisi secondo la loro funzione. Alla fine di questo stesso capitolo è riportato anche un sommario dei comandi.

Le descrizioni seguenti usano le barre (/) per indicare i caratteri delimitatori (cioè quelli che racchiudono le stringhe). I nomi dei comandi appaiono in maiuscolo; i tipi degli argomenti sono in minuscolo e appaiono come indicato nella seguente tavola.

Notazione	Descrizione
a,b	numeri di linea (oppure "." o "*")
gc	gruppo di comandi
m,n	numeri
q	lista di qualificatori (anche vuota)
er	espressione di ricerca
s,t	stringhe di caratteri arbitrari
sw	valore di switch (+ o -)
/	delimitatore di stringa

Tavola 4-1: notazione per la descrizione dei comandi

Nota: *le descrizioni dei comandi che appaiono nel resto di questo manuale con la notazione appena illustrata mostrano la SINTASSI del comando; non sono esempi di quello che si digita effettivamente. Gli esempi appaiono sempre in*

```
questo tipo di carattere
```

4.2.3.1 Selezione di una linea corrente

Questi comandi non hanno altra funzione che quella di selezionare una nuova linea corrente. EDIT aggiunge le linee che ha già trattato nella coda di output (per ulteriori dettagli sulla coda di output, vedere la sezione 4.1, "Introduzione a EDIT"). EDIT recupera dalla coda di output le linee che necessitano di nuove elaborazioni per poi aggiungerle nuovamente. M accetta un numero di linea, un punto o un asterisco. Quindi, usando la notazione dei comandi descritta in precedenza, la sintassi corretta di M è la seguente:

Ma

dove Ma si muove, in avanti o indietro, alla linea "a" del file sorgente. Si può accedere per numero di linea solamente a linee originali.

M+

rende corrente l'ultima linea effettivamente letta dal file. M+ si muove attraverso tutte le linee in memoria finché non raggiunge l'ultima.

M-

rende corrente l'ultima linea della coda di output. In pratica è come comunicare a EDIT di muovere l'indicatore della linea corrente più indietro possibile.

N

si muove in avanti alla linea successiva del sorgente. Quando la linea corrente è l'ultima linea del file sorgente, l'esecuzione di un comando N non genera un errore. EDIT incrementa il numero di linea aggiungendo un 1 e creando una linea speciale di end-of-file. Comunque, se si cerca di usare un comando N quando ci si trova già nella linea finale del file sorgente, EDIT restituisce un errore.

P

si sposta alla linea precedente. È possibile spostarsi all'indietro di diverse linee sia ripetendo più volte P, sia anteponendo a esso un numero. Il numero che si indica deve essere uguale al numero di linee che si desidera far scorrere.

La sintassi del comando F (trova) è

F er

F ricerca la linea indicata con l'espressione "er". La ricerca ha inizio alla linea corrente e prosegue in avanti attraverso tutto il file. Per mantenere la posizione raggiunta per effetto dei comandi precedenti, come una cancellazione di linea, la ricerca viene svolta iniziando dalla linea corrente. Un comando F senza argomenti opera la ricerca usando l'ultima espressione specificata.

La sintassi del comando BF (ricerca all'indietro) è

BF er

BF si comporta nello stesso modo di F, a parte il fatto che opera in direzione contraria, partendo dalla linea corrente e proseguendo fino a che non trova una linea che soddisfi l'espressione di ricerca.

4.2.3.2 Inserimento e cancellazione di linee

I comandi possono selezionare una nuova linea come effetto collaterale della loro funzione principale. Quelli che comportano l'inserimento di testo nelle linee devono essere gli ultimi comandi su una linea. Il testo da inserire si trova in linee successive, l'ultima delle quali deve contenere solo una Z.

È possibile usare il comando Z per cambiare il terminatore. EDIT riconosce il terminatore sia in maiuscolo che in minuscolo. Per esempio, usando la consueta notazione,

```
Ia  }   <inserimento del testo, tante linee
Z   }   quante ne sono necessarie>
```

inserisce il testo prima di "a". Ricordate che "a" può essere un numero di linea specifico, un punto (che rappresenta la linea corrente) o un asterisco (che rappresenta l'ultima linea del file sorgente). Se si omette "a", EDIT inserisce il testo prima della linea corrente; altrimenti la linea "a" diventa quella corrente.

I/s/

inserisce il contenuto del file "s" (ricordate che "s" significa una stringa qualunque) prima della linea corrente.

```
Ra b }   <testo di sostituzione>
Z    }
```

Ra b/s/

Il comando R equivale a D seguito da I. Il secondo numero di linea deve essere maggiore o uguale al primo. Si può omettere il secondo numero se si desidera sostituire solamente una linea (cioè se b = a). Si possono omettere entrambi i numeri se si vuole sostituire solo la linea corrente. La linea che segue b diventa la nuova linea corrente.

La sintassi del comando D (Delete, cancella) è la seguente:

Da b

D cancella tutte le linee tra a e b comprese. Si può omettere il secondo numero di linea se si desidera cancellare solamente una linea (cioè se b = a). Si possono omettere entrambi i numeri se si vuole cancellare la sola linea corrente. La linea che segue b diventa la nuova linea corrente.

La sintassi del comando DF (Delete Find, cancella e ricerca) è

DF er

Il comando DF comunica a EDIT di cancellare le linee successive al sorgente finché non incontra una linea che soddisfi l'espressione di ricerca.

Questa linea diventa poi quella corrente. Un comando DF senza argomenti usa l'ultima espressione di ricerca indicata, e mentre esegue la ricerca cancella le linee.

4.2.4 Porzioni di linea

EDIT agisce solitamente su una linea completa. Comunque, si possono definire parti della linea, chiamate porzioni di linea, sulle quali EDIT può eseguire i comandi successivi. Questa sezione descrive i comandi necessari per definire una porzione.

4.2.4.1 La porzione operativa

EDIT di solito scandisce tutti i caratteri di una linea mentre cerca una data stringa. Comunque è possibile definire una “porzione di linea” in modo che la ricerca parta dall'inizio di tale porzione e non dall'inizio della linea. In tutte le descrizioni dei comandi di EDIT, “l'inizio della linea” significa sempre “l'inizio della porzione operativa”.

Ogni volta che EDIT verifica la linea corrente, indica la posizione della porzione operativa visualizzando un carattere “>” direttamente sotto la linea. Per esempio nel seguente caso:

```
26.
Questa e' la linea 26 e' questa
          >
```

la porzione operativa contiene i caratteri alla destra del simbolo: “la linea 26 è questa”. Se l'indicatore si trova all'inizio della linea, EDIT lo omette.

Il margine sinistro della porzione è anche chiamato **puntatore al carattere** e per muoverlo sono disponibili i seguenti comandi:

>

muove il puntatore di un carattere a destra,

<

muove il puntatore di un carattere a sinistra,

PR

porta il puntatore all'inizio della linea.

La sintassi del comando PA (Point After, punta dopo) è

PA q/s/

Il comando aggiorna il puntatore in modo che il primo carattere della

porzione sia il primo carattere che segue la stringa s. Per esempio,

PA L//

muove il puntatore alla fine della linea.
La sintassi del comando PB (Point Before, punta prima) è

PB q/s/

Il comando PB è uguale a PA, ma include nella porzione la stringa stessa.

4.2.4.2 Operazioni sui singoli caratteri della linea corrente
I due comandi seguenti muovono il cursore di una posizione verso destra dopo aver forzato il cambiamento in maiuscolo o minuscolo della lettera sopra la quale si trovano. Se il carattere non è una lettera o se è già nella forma corretta, questi comandi sono equivalenti a >.
Il comando

$

forza il minuscolo.
Il comando

%

forza il maiuscolo.
Il comando "_" (underscore, trattino alla base della linea) cambia il primo carattere della porzione in uno spazio, muovendo poi il cursore di una posizione verso destra.
Il comando

#

cancella il primo carattere della porzione. Il resto della porzione si muove di una posizione verso sinistra, lasciando che il cursore punti al successivo carattere nella linea. Il comando equivale esattamente a

E/s//

dove "s" è il primo carattere nella porzione. Per ripetere l'effetto, si specifica un numero prima del comando "#". Per esempio,

5#

cancella i successivi cinque caratteri della porzione. Se si usa un numero uguale o maggiore al numero di caratteri contenuti nella porzione, viene cancellata tutta la porzione. EDIT tratta una sequenza di comandi "#" nello stesso modo di un singolo, comando "#" ripetuto più volte. Quindi ##### equivale a digitare un singolo # seguito da RETURN per cinque volte.

È possibile usare una combinazione di comandi >, %, $, _ e # per trattare una linea carattere per carattere, con i comandi che appaiono sotto al carattere sul quale agiscono. Il testo e i comandi seguenti spiegano come questo avviene:

```
o Taciturno,, Vieni eeCammina con   noi
%>$$$$$$$$$#>>$$$$$>$_$$$$$$$>$$$##
```

I comandi nell'esempio precedente modificano la linea in

```
O taciturno, vieni e cammina con noi
```

lasciando il cursore immediatamente prima della parola "noi".

4.2.5 Operazioni sulle stringhe nella linea corrente

Per indicare quale parte della linea corrente è da considerare, si può alterare la stringa iniziale, oppure spostare il puntatore a una variante della stessa, come descrivono le due sezioni successive.

4.2.5.1 Operazioni principali sulle stringhe

Sono disponibili tre comandi simili per alterare parti della linea corrente. I comandi A, B ed E inseriscono il loro secondo argomento (stringa) rispettivamente dopo, prima e operando una sostituzione con il primo argomento. Se la linea corrente è

```
Il Falegname parlo'
```

allora i comandi

```
E U/falegname/Carpentiere/    <sostituisce>
B/par/non /                   <inserisce la stringa prima>
A L//;/                       <inserisce la stringa dopo>
```

cambieranno la stringa in

```
Il Carpentiere non parlo';
```

4.2.5.2 La stringa nulla

Si può usare la stringa nulla (//), detta anche vuota, con ogni comando di stringa. Se si usa la stringa nulla come secondo argomento di un comando E, EDIT rimuove la prima stringa dalla linea. Un comando A o B con la seconda stringa vuota non modifica nulla. Una stringa nulla posta come primo argomento in uno dei tre comandi, corrisponde al punto di partenza della ricerca, il quale, in mancanza dei qualificatori E e L che indicano la fine della linea, corrisponde alla posizione corrente del puntatore (generalmente l'inizio della linea). Per esempio,

```
A//carpentiere/
```

colloca il testo "carpentiere" prima di niente, cioè all'inizio della linea. Mentre

```
A L//carpentiere/
```

inserisce "carpentiere" alla fine della linea dopo l'ultima volta che si incontra la stringa nulla.

4.2.5.3 Variazioni con spostamento del puntatore

I comandi AP (insert After and Point, inserisci dopo e sposta il puntatore), BP (insert Before and Point, inserisci prima e sposta il puntatore) ed EP (Exchange and Point, sostituisci e sposta il puntatore) accettano due stringhe come argomenti e agiscono esattamente come A, B ed E. Però AP, BP ed EP hanno una caratteristica in più: quando l'operazione è stata completata, il puntatore viene spostato in modo che venga a trovarsi a sinistra del primo carattere che segue entrambe le stringhe. Quindi, usando la consueta notazione sintattica,

AP/s/t/

equivale a

A/s/t/;PA/st/

mentre

BP/s/t/

equivale a

B/s/t/;PA/ts/

e

```
2EP U/giov/Giov/
```

cambierà

```
giovanni e GIOvannino
```

in

```
Giovanni e Giovannino
```

lasciando il cursore esattamente in corrispondenza della v di Giovannino.

4.2.5.4 Cancellazione di parti della linea corrente

I comandi DTA (Delete Till After, cancella fino a dopo - cioè stringa compresa) e DTB (Delete Till Before, cancella fino a prima - cioè stringa esclusa) si usano per cancellare dall'inizio della linea (o dalla posizione del puntatore) fino alla stringa indicata. Per cancellare da una certa stringa fino al termine della linea si usano i comandi DFA (Delete From After, cancella da dopo - cioè partendo dalla stringa, stringa esclusa) e DFB (Delete From Before, cancella da prima - cioè cancella partendo dalla stringa, stringa compresa). Se la linea corrente è

```
Tutti i Re hanno cavalli e tutti i Re hanno uomini
```

allora il comando

```
DTB L/i Re/
```

la modificherà in

```
i Re hanno uomini
```

mentre

```
DTA/cavalli /
```

la cambierà in

```
e tutti i Re hanno uomini
```

4.2.6 Miscellanea di comandi per la linea corrente

Questa sezione spiega come ripetere i comandi che coinvolgono le stringhe, come dividere la linea corrente e come riunire le linee.

Ogni volta che EDIT esegue un comando che altera una stringa (per esempio A, B o E), tale comando diventa quello corrente di alterazione delle stringhe. Per ripeterlo si può digitare un singolo apostrofo ('). Il comando ' non ha argomenti, visto che usa quelli dell'ultimo comando A, B o E.

ATTENZIONE: se si usano sequenze come la seguente si possono verificare effetti inaspettati:

```
E/torre/cavallo/; 4('; E/pedone/regina/)
```

La seconda esecuzione del comando ' e le successive si riferiscono a un diverso comando rispetto alla prima. L'esempio precedente scambia due volte torre con cavallo e sette volte pedone con regina, invece di operare un singolo scambio di torre con cavallo e poi quattro scambi di torre con cavallo e pedone con regina.

4.2.6.1 Divisione e unione di linee

EDIT è essenzialmente un editor di linea. La maggior parte dei comandi di EDIT non opera su più linee, ma questa sezione descrive i comandi per dividere una linea in più d'una e per unire due o più linee successive.

Per dividere una linea prima di una determinata stringa si usa il comando SB, la cui sintassi è

SB q/s/

SB accetta un qualificatore opzionale rappresentato da q, e una stringa /s/. SB divide la linea corrente prima del contesto indicato dal qualificatore e dalla stringa. EDIT manda la prima parte della linea nell'output e trasforma il resto in una nuova linea corrente non originale.

Per dividere una linea dopo un determinato contesto si usa il comando SA, la cui sintassi è

SA q/s/

SA accetta un qualificatore opzionale e una stringa (q e /s/). Questo comando divide la linea corrente dopo il contesto specificato dal qualificatore e dalla stringa.

Per concatenare una linea alla successiva si usa il comando CL, la cui sintassi è

CL/s/

CL accetta una stringa opzionale rappresentata da /s/. Questo comando forma una nuova linea concatenando nell'ordine la linea corrente, la stringa indicata e la successiva linea del file sorgente. Se la stringa è nulla, si può usare il comando CL senza indicare alcuna stringa.

Come esempio sulla divisione e unione di linee, si consideri il testo

```
Humpty Dumpty sedeva su un muretto; Humpty
Dumpty fece un
grande voletto.
```

I versi appaiono discontinui, le righe devono essere pareggiate. Facendo sì che la prima linea sia quella corrente, i comandi

```
SA /; /; 2CL/ /
```

cambieranno il verso in

```
Humpty Dumpty sedeva su un muretto;
```

lasciando

```
Humpty Dumpty fece un grande voletto.
```

come nuova linea corrente.

4.2.7 Ispezione di parte del sorgente: il comando Type

Tutti i comandi che seguono comunicano a EDIT di scorrere il file sorgente, mandando le linee al file di verifica così come al normale file di output. Poiché questi comandi sono usati per la maggior parte in modo interattivo (cioè con verifica sullo schermo), sono anche conosciuti come comandi "type". Possono infatti essere usati per visualizzare (in inglese "type") sullo schermo le linee indicate dall'utente. Questo non vuol dire che non si possano usare per dirigere l'output in un file. Dopo che EDIT ha eseguito uno di questi comandi,

l'ultima linea scritta (cioè visualizzata) diventa la nuova linea corrrente.
La sintassi del comando T (dall'inglese Type) è

Tn

Tn visualizza "n" linee. Se "n" viene omesso, la visualizzazione prosegue fino al termine del file sorgente. Comunque si può sempre interrompere l'operazione di scrittura con CTRL-C.

Nota: *in tutto questo manuale, quando compare un trattino tra due tasti, significa che bisogna premerli nello stesso momento. Quindi CTRL-C significa tenere premuto il tasto CTRL e nel contempo il tasto C.*

Quando si usa il comando T, la prima linea che EDIT scrive è quella corrente, quindi, per esempio

```
F /la mia invenzione/; T6
```

scrive sei linee partendo da quella che contiene "la mia invenzione".
Il comando

TP

scrive le linee contenute nella coda di output. Quindi TP è equivalente all'esecuzione di M- seguito da T, finché non si raggiunge l'ultima linea attualmente letta dal sorgente.
Il comando

TN

scrive finché EDIT non ha cambiato tutte le linee nella coda di output (per maggiori informazioni sulla coda di output, vedere la sezione 4.1, "Introduzione a EDIT"). Quindi un comando TN scrive N linee, dove N è il numero specificato con l'opzione P (per maggiori informazioni sull'opzione P, vedere la sezione 4.1.1, "Attivazione di EDIT"). Il vantaggio del comando TN è che qualunque linea visibile durante l'operazione di scrittura è disponibile in memoria per i comandi P e BF.
La sintassi del comando TL (scrittura con numeri di linea) è la seguente:

TLn

TLn scrive n linee come T, ma con l'aggiunta dei numeri di linea. Le linee non originali non possiedono numeri di linea e EDIT utilizza per loro un

"++++" al posto del numero. Per esempio,

```
20 Buongiorno a tutti quanti
++++oggi e' una bella giornata
```

La linea originale che inizia con "Buongiorno" ha un numero di linea. Quella non originale, inserita dopo la linea 20, inizia con ++++ (ricordatevi che si può usare il comando = per numerare nuovamente le linee, comprese quelle non originali).

4.2.8 Controllo dei comandi e dei file di input e output

EDIT usa quattro tipi di file:

- file di comandi
- file di input
- file di output
- file di verifica

Una volta che EDIT è stato attivato, non si può cambiare il file di verifica per mezzo di un comando (per maggiori informazioni sul file di verifica, riferirsi alla sezione 4.1.1, "Attivazione di EDIT"). Le seguenti sezioni descrivono i comandi che possono cambiare i file di comandi, di input e di output impostati quando si è attivato EDIT.

4.2.8.1 File di comandi

EDIT legge i comandi dal terminale oppure dal file che è stato indicato con WITH. Per far leggere i comandi da un altro file si usa il comando C, la cui sintassi è

C .s.

dove la stringa "s" rappresenta il nome del file. Dal momento che l'AmigaDOS usa le barre per separare i filename, si devono usare dei punti (.) o qualche altro simbolo, per delimitare il nome di un file. Un simbolo presente in una stringa non può essere usato per delimitarla. Quando EDIT ha terminato di eseguire tutti i comandi di un file (o quando si impartisce il comando Q), lo chiude e restituisce il controllo al comando che segue il comando C. Per esempio, il comando

```
C .:T/XYZ.
```

legge ed esegue i comandi contenuti nel file ":T/XYZ".

4.2.8.2 File di input

Per inserire l'intero contenuto di un file in un punto particolare del sorgente, si possono usare i comandi I e R. Questi comandi sono stati descritti in precedenza nella sezione 4.1.2.7.

La sezione 4.1.1 descrive come si attiva EDIT e in essa ci si riferisce al file sorgente come al file FROM. Comunque si possono associare al file FROM altri file, per mezzo del comando FROM. Il comando FROM ha il seguente formato:

FROM .s.

dove la stringa "s" è il nome del file. Un comando FROM senza argomenti seleziona nuovamente il file sorgente originale.

Quando EDIT esegue un comando FROM, la linea corrente rimane tale, mentre la linea successiva proviene dal nuovo file sorgente.

EDIT non chiude un file sorgente quando quest'ultimo cessa di essere quello corrente; si possono leggere da esso ulteriori linee selezionandolo nuovamente in seguito.

Per chiudere un file di output che è stato aperto con EDIT e che in seguito si desidera aprire per input (o per qualunque altro motivo) si deve usare il comando CF (Close File, chiudi il file). Il comando CF ha il seguente formato:

CF .s.

dove la stringa "s" rappresenta il nome del file. Quando si termina una sessione di EDIT, vengono chiusi automaticamente tutti i file aperti.

Nota: *ogni volta che si apre un file, EDIT lo elabora dall'inizio. Se si chiude un file con CF, EDIT inizia con la prima linea del file se viene nuovamente aperto e non con la linea corrente al momento della chiusura.*

Qui di seguito è riportato un esempio dell'uso del comando FROM per unire due file.

Comando	**Azione**
M10	*Scorre le linee 1-9 dal file FROM (sorgente)*
FROM .XYZ.	*Seleziona un nuovo input, la linea 10 rimane quella corrente*
M6	*Scorre la linea 10 da FROM e le linee 1-5 da XYZ*
FROM	*Seleziona nuovamente FROM*
M14	*Scorre la linea 6 da XYZ e le linee 11-13 da FROM*
FROM .XYZ.	*Seleziona nuovamente XYZ*

M*	*Scorre la linea 14 da FROM e il resto di XYZ*
FROM	*Seleziona nuovamente FROM*
CF .XYZ.	*Chiude il file XYZ*
M*	*Scorre il resto di FROM (dalla linea 15 fino alla fine del file)*

4.2.8.3 File di output

EDIT solitamente manda l'output al file con il nome TO. EDIT, comunque, non invia l'output immediatamente ma mantiene, il più a lungo possibile, un certo numero di linee in una coda nella memoria centrale. Queste linee sono le precedenti linee correnti o le linee che sono state superate per raggiungere la linea corrente attuale. Il numero di linee che EDIT può mantenere dipende dalle opzioni che sono state indicate quando lo si è richiamato. Dal momento che EDIT mantiene in memoria queste linee, ha la capacità di scorrere all'indietro il file sorgente.

Per associare la coda di output con un file diverso da quello che ha come filename TO, si può usare il comando TO. Questo comando ha il formato

TO .s.

dove "s" è il nome del file.

Quando EDIT esegue un comando TO, che indica un diverso file di output, provvede a trasferirvi le linee ancora esistenti nella coda.

EDIT non chiude un file di output quando quest'ultimo smette di essere il file corrente; selezionandolo nuovamente si possono aggiungere ulteriori linee. L'esempio seguente mostra come si può dividere il file sorgente tra il file destinazione principale TO e un altro a piacere XYZ.

Comando	**Azione**
M11	*Scorre le linee 1-10 in TO*
TO .XYZ.	*Cambia il file di output*
M21	*Scorre le linee 11-20 in XYZ*
TO	
M31	*Scorre le linee 21-30 in TO*
TO .XYZ.	
M41	*Scorre le linee 31-40 in XYZ*
TO	

Se si desidera usare nuovamente un file, occorre chiuderlo esplicitamente. Il comando

CF .filename.

chiude il file che ha il nome indicato come argomento.

Questi comandi di input/output sono utili quando si desidera muovere in un punto più avanti nell'output una parte del file sorgente. Per esempio,

Comando	**Azione**
TO .:T/1.	*Output in un file temporaneo*
1000N	*Avanza attraverso il sorgente*
TO	*Ritorna a TO*
CF .:T/1.	*Chiude il file di output :T/1*
I2000.:T/1.	*Lo usa nuovamente come file di input*

Se si usa il comando CF per i file con i quali non si lavorerà più, si minimizza la quantità di memoria necessaria.

4.2.9 Iterazioni (Loop)

È possibile aggiungere un numero decimale senza segno prima di diversi comandi per indicarne l'iterazione, per esempio

```
24N
```

È anche possibile indicare il numero di ripetizioni di un gruppo di comandi, come avviene per i comandi individuali. Per esempio

```
12(F/mano/; E/mano/dito/; 3N)
```

Se si fornisce un valore di ripetizione zero (0), il comando o il gruppo di comandi sono ripetuti infinitamente finché EDIT non raggiunge la fine del sorgente.

4.2.10 Operazioni globali

Le operazioni globali sono operazioni che hanno luogo automaticamente mentre EDIT scandisce il file sorgente verso la fine. Si possono iniziare e terminare le operazioni globali con alcuni comandi speciali, che verranno illustrati nelle prossime sezioni.

> **ATTENZIONE:** occorre stare molto attenti a non lasciare attivo o "abilitato" qualche comando globale quando si scorre il sorgente all'indietro: si potrebbe perdere il frutto di molto lavoro!

4.2.10.1 Impostazione delle modifiche globali

I tre comandi GA, GB e GE sono studiati per operare semplici modifiche di stringa in ogni linea. La loro sintassi è la seguente:

GA q/s/t/
GB q/s/t/
GE q/s/t/

Questi comandi applicano un comando A, B o E, a seconda del caso, a ogni ricorrenza della stringa "s" in una nuova linea corrente e anche nella linea che è corrente al momento dell'esecuzione del comando.

I comandi G non scandiscono nuovamente il testo appena sostituito; per esempio il comando

```
GE/tigre bianca/tigre bianca/
```

non ripete in eterno la stessa operazione, anche se non produce alcuna alterazione nel testo. Comunque, come risultato di tale "modifica", EDIT dovrebbe verificare certe linee.

EDIT applica le modifiche globali a ogni nuova linea corrente seguendo l'ordine nel quale si impartiscono i comandi.

4.2.10.2 Disattivazione delle modifiche globali

Il comando REWIND disattiva tutte le operazioni globali. Si può usare in qualunque momento il comando CG (Cancel Global, cancella il comando globale) per disattivare comandi globali singolarmente.

Quando un'operazione globale è impostata per mezzo di uno dei comandi GA, GB o GE, a essa viene assegnato un numero di identificazione (per esempio G1) che viene trasferito nel file di verifica. L'argomento di CG è il numero dell'operazione globale che si vuole disattivare. Se CG viene eseguito senza argomenti, EDIT disattiva tutti i comandi globali.

4.2.10.3 Sospensione delle modifiche globali

Si possono sospendere, e in seguito riprendere, le operazioni globali singolarmente per mezzo dei comandi DG (Disable Global, disabilita globale) ed EG (Enable Global, abilita globale) che ricevono come argomento il numero di identificazione. Se si omette l'argomento, tutti i comandi globali vengono attivati o disattivati, a seconda del comando usato.

4.2.11 Visualizzazione dello stato del programma

Due comandi che iniziano con SH (SHow, mostra) visualizzano informazioni riguardanti lo stato di EDIT e del file di verifica.

Il comando SHD (SHow Data, mostra i dati) ha il formato

SHD

e visualizza i valori delle ultime informazioni salvate, come l'ultima espressione di ricerca.

Il comando SHG (SHow Global, mostra i globali) ha il formato

SHG

e visualizza i comandi globali correnti, insieme con i loro numeri di identificazione. Dice anche quante volte ogni espressione di ricerca globale trova corrispondenze.

4.2.12 Termine dell'esecuzione di EDIT

Per porre fine all'esecuzione di EDIT senza raggiungere la fine del file sorgente si usa il comando W (Windup, esci salvando). Il comando W non può essere usato se in quel momento l'output non è diretto verso il file TO. EDIT, quando ha raggiunto la fine del file, termina l'esecuzione chiudendo tutti i file e liberando la memoria occupata. Raggiungendo la fine del file di comandi di livello più elevato, si ottiene lo stesso effetto del comando W. EDIT, se è stato attivato indicando solo il filename FROM, sostituisce il nome del file temporaneo di output che aveva creato con quello del file originale (cioè il file FROM), e cambia il nome del file che contiene le informazioni originali in ":T/EDIT-BACKUP". Questo file di backup è naturalmente disponibile finché EDIT non viene eseguito nuovamente.

Il comando STOP ferma EDIT immediatamente, bloccando ogni output o input successivo. In particolare il comando STOP evita che EDIT scriva sopra il file sorgente originale. Digitare STOP assicura che non sia effettuato nessun cambiamento alle informazioni di input.

Il comando Q evita che EDIT continui a eseguire il file di comandi corrente (EDIT inizialmente accetta i comandi dalla tastiera, ma si può indicare un file di comandi per mezzo della keyword WITH o del comando C) e rende attivo quello precedente. Il comando Q impartito al livello più esterno ha lo stesso effetto di un W.

4.2.13 Verifica della linea corrente

Le seguenti circostanze possono far sì che venga effettuata una verifica automatica.

- Quando si impartisce una sequenza di comandi per una linea corrente che EDIT non ha verificato da quando è diventata corrente, oppure che è stata alterata dopo l'ultima verifica.
- Quando EDIT si muove oltre una linea che è stata alterata, ma non ancora verificata.
- Quando EDIT visualizza un messaggio d'errore.

Nei primi due casi, si determina una verifica se lo switch V è attivato. Il comando

V sw

cambia lo stato dello switch V. Lo switch è attivato (V+) se lo stato iniziale di EDIT è interattivo (i comandi e le verifiche sono connessi con il terminale), altrimenti è disattivato (V-).

Per richiedere esplicitamente la verifica della linea corrente si usa il comando seguente:

?

Questo comando verifica la linea corrente. Viene eseguito automaticamente se lo switch V è attivato e se le informazioni nella linea sono state cambiate. La verifica consiste nel numero di linea (oppure ++++ se la linea non è originale), seguito dal testo nella linea successiva.

Una forma alternativa della verifica, utile per le linee contenenti caratteri non stampabili, è fornita dal comando

!

Il comando ! verifica la linea corrente con gli indicatori dei caratteri. EDIT produce due linee di verifica. La prima è la linea corrente nella quale EDIT sostituisce tutti i caratteri non grafici con il primo carattere del loro valore esadecimale. Nella seconda linea, EDIT visualizza un segno meno sotto ogni lettera maiuscola e la seconda cifra esadecimale nelle posizioni corrispondenti a caratteri non grafici. Le altre posizioni contengono degli spazi.

L'esempio seguente usa i comandi ? e !. Per verificare la linea corrente si usa il comando ?. Se, per esempio, quando si usa il comando ? appaiono le scritte:

```
?
1.
Il Tricheco e il ??
```

allora si può cercare di usare il comando E per cambiare "??" in "Carpentiere". Comunque, EDIT potrebbe non riconoscere il testo visualizzato con "??" come due punti di domanda se i caratteri "??" corrispondono a due caratteri non stampabili. Per scoprire che cosa ci sia veramente si usa il comando !:

```
!
1.
Il Tricheco e il 11
- -              44
```

In questo caso i due caratteri fittizi "??" corrispondono entrambi al codice esadecimale $14, che è un carattere non stampabile. Per correggere la linea, si possono usare il puntatore al carattere e il comando #, cancellando i caratteri spuri prima di inserire il testo corretto (per ulteriori dettagli sul puntatore e il comando #, riferirsi alla sezione 4.2.4, "Porzioni di linea").

4.2.14 Miscellanea di comandi

Questa sezione descrive tutti i comandi che non appartengono esattamente a nessuna delle categorie precedenti. Vi si descrive come cambiare il carattere terminatore dell'inserimento, come disabilitare gli spazi di trailing, come rinumerare le linee e come tornare all'inizio del file sorgente.

Per cambiare il carattere di fine inserimento per il testo, si usa il comando Z. Il comando Z ha la seguente forma:

Z/s/

dove /s/ rappresenta una stringa. La stringa può essere lunga fino a 16 caratteri e non sono significative le differenze tra maiuscolo e minuscolo. Infatti la ricerca per la stringa di fine inserimento (di default "Z") viene effettuata usando i qualificatori PU.

Per abilitare o disabilitare gli spazi di trailing si usa il comando TR, che

ha la forma seguente:

TR sw

dove sw rappresenta uno switch (+ per attivare, - per disattivare). EDIT di solito sopprime tutti gli spazi di trailing. TR+ permette la presenza degli spazi di traling sia nelle linee di input sia in quelle di output.

Per numerare nuovamente le linee del sorgente, si usa il comando =, che ha il seguente formato:

=n

dove "n" rappresenta un numero. Il comando =n assegna il numero n alla linea corrente. Muovendosi lungo le linee successive, EDIT rinumera sia quelle che sono originali sia quelle che non lo sono. Però, se ci si muove lungo linee precedenti dopo aver usato il comando =, EDIT indica tutte le linee contenute nella coda come non originali. Quando si "riavvolge" il file sorgente, EDIT rinumera tutte le linee: originali, non originali e quelle rinumerate con il comando =.

Per "riavvolgere" (cioè rileggere dall'inizio) il file sorgente si usa il comando REWIND. Per esempio, scrivendo

```
REWIND
```

si "riavvolge" il file di input in modo che la linea 1 sia nuovamente la linea corrente. Per prima cosa EDIT scandisce il resto del file sorgente (per comandi globali e così via). Poi scrive le linee nel file destinazione, che viene in seguito chiuso e poi riaperto come nuovo sorgente. Il file sorgente originario viene chiuso usando un file temporaneo come destinazione. Qualunque comando globale attivo viene cancellato. EDIT non richiede che sia indicata la parola completa (cioè REWIND); è sufficiente digitare REWI oppure REWIN.

4.2.15 Abbandono dell'editing interattivo

Per arrestare l'esecuzione di un comando, quasi sempre si preme CTRL-C. In particolare, se ci si accorge che un'espressione di ricerca è errata, l'uso di CTRL-C interrompe la ricerca di un testo non voluto. Il comando T visualizza il resto del file sorgente fino al termine e, in modo simile a quanto si è visto in precedenza, la pressione di CTRL-C ne ferma l'esecuzione.

Dopo aver premuto CTRL-C, EDIT risponde con il messaggio

```
*** BREAK
```

e ritorna a leggere i comandi. La linea corrente dipende, naturalmente, dal momento esatto nel quale si è premuto CTRL-C.

4.3 Tavola di consultazione rapida

Questo elenco usa le seguenti abbreviazioni:

Notazione	**Descrizione**
qs	Stringa qualificata
t	Stringa
n	Numero di linea, oppure . o * (la linea corrente e l'ultima)
sw	+ oppure - (attivato o disattivato)

Comandi di gestione del puntatore al carattere (comandi per la porzione di linea)

Comando	**Azione**
<	Muove il puntatore a sinistra
>	Muove il puntatore a destra
#	Cancella il carattere indicato dal puntatore
$	Fa diventare minuscolo il carattere indicato dal puntatore
%	Fa diventare maiuscolo il carattere indicato dal puntatore
_	Cambia il carattere indicato dal puntatore in uno spazio
PA qs	Muove il puntatore dopo qs
PB qs	Muove il puntatore prima di qs
PR	Muove il puntatore all'inizio della linea

Comandi di posizionamento

Comando	**Azione**
M n	Muove alla linea n
M +	Muove alla linea più alta in memoria
M -	Muove alla linea più bassa in memoria
N	Linea successiva
P	Linea precedente
REWIND	Rilegge il file di input dall'inizio

Comandi di ricerca

Comando	**Azione**
F qs	Ricerca la stringa qs
BF qs	Uguale a F, ma si muove all'indietro nel file
DF qs	Uguale a F, ma cancella le linee che incontra

Verifica del testo

Comando	**Azione**
?	Verifica la linea corrente
!	Verifica usando gli indicatori dei caratteri
T	Stampa fino al termine del file
T n	Stampa n linee
TL n	Stampa n linee con i numeri di linea
TN	Stampa finché non cambia il buffer
TP	M-, poi stampa fino all'ultima linea nel buffer
V sw	Abilita o disabilita la verifica

Operazioni sulla linea corrente

Comando	**Azione**
A qs t	Colloca la stringa t dopo qs
AP qs t	Uguale ad A, ma sposta il puntatore
B qs t	Colloca la stringa t prima di qs
BP qs t	Uguale a B, ma sposta il puntatore
CL t	Concatena la linea corrente, t e la linea successiva
D	Cancella la linea corrente
DFA qs	Cancella da dopo qs fino al termine della linea
DFB qs	Cancella da prima di qs fino al termine della linea
DTA qs	Cancella dall'inizio della linea fino a qs compresa
DTB qs	Cancella dall'inizio della linea fino a qs esclusa
E qs t	Cambia la stringa qs con t
EP qs t	Uguale a E, ma muove il puntatore
I	Inserisce un testo proveniente dal terminale prima della linea
I t	Inserisce dal file t
R	Sostituisce il testo dal terminale
R t	Sostituisce il testo dal file t
SA qs	Divide la linea dopo qs
SB qs	Divide la linea prima di qs

Comandi globali

Comando	**Azione**
GA qs t	In modo globale, pone t dopo qs
GB qs t	In modo globale, pone t prima di qs
GE qs t	In modo globale, cambia qs con t
CG n	Disattiva il comando globale n (tutti, se n è omesso)
DG n	Sospende il comando globale n (tutti, se n è omesso)
EG n	Riprende il comando globale n (tutti, se n è omesso)
SHG	Visualizza informazioni sui comandi globali usati

Manipolazione dell'Input/Output

Comando	**Azione**
FROM	Riceve le linee dal file originale
FROM t	Riceve le linee dal file t
TO	Direziona l'output nel file destinazione
TO t	Direziona l'output nel file t
CF t	Chiude il file t

Altri comandi

Comando	**Azione**
'	Ripete il precedente comando A, B o E
= n	Imposta il numero di linea a n
C t	Preleva i comandi dal file t
H n	Definisce l'alt alla linea n. Se n=* l'alt viene rimosso
Q	Esce dal livello di comando; uguale a W se il livello è 1
SHD	Visualizza informazioni
STOP	Stop
TR sw	Abilita e disabilita la rimozione degli spazi di trailing
W	Termine delle operazioni
Z t	Imposta il terminatore dell'input alla stringa t

Appendice

Codici e messaggi d'errore

I messaggi d'errore che appaiono sullo schermo quando si usano i comandi FAULT o WHY, rientrano in due categorie generali:

- errori dell'utente
- errori del programmatore

Questa appendice fornisce la causa probabile di ciascun tipo d'errore e un suggerimento per cercare di eliminarli. I messaggi d'errore sono stati suddivisi nelle due categorie e ordinati secondo il loro numero di codice.

Errori dell'utente

103: insufficient free store (memoria libera insufficiente)
Causa probabile:
Non c'è abbastanza memoria fisica nell'Amiga per portare a termine questa operazione.
Suggerimenti per l'eliminazione:
Per prima cosa cercate di fermare le applicazioni in esecuzione di cui non avete bisogno: per esempio chiudete tutte le finestre non necessarie. Altrimenti, acquistate dell'altra memoria. Oppure fermate i processi meno importanti e impartite di nuovo il comando. Può darsi che ora abbiate abbastanza memoria, ma che sia diventata "frammentata": un nuovo boot potrebbe esservi d'aiuto.

104: task table full (tavola dei task piena)
Causa probabile:
È stato superato il limite di 20 task, o l'equivalente, del CLI.

120: argument line invalid or too long (argomenti della linea non validi oppure troppo lunghi)
Causa probabile:
Gli argomenti per questo comando non sono corretti oppure contengono troppe opzioni.
Suggerimenti per l'eliminazione:
Consultate le spiegazioni dei comandi nel capitolo 2 di questo manuale per scrivere gli argomenti nel formato corretto.

121: file is not an object module (il file non è un modulo oggetto)
Causa probabile:
O è errato il nome del comando, oppure il file non è in un formato caricabile.
Suggerimenti per l'eliminazione:
È sufficiente riscrivere il nome del file oppure assicurarsi che il file sia un programma eseguibile. È necessario ricordarsi che per eseguire una sequenza di comandi si deve anteporre il comando EXECUTE al nome del file.

122: invalid resident library during load (libreria residente non valida durante il caricamento)

202: object in use (oggetto in uso)
Causa probabile:
Il file o la directory indicata è già usato da un'altra applicazione in modo non compatibile con l'operazione richiesta.
Suggerimenti per l'eliminazione:
Se un'applicazione sta scrivendo in un file, nessun'altra può accedervi in lettura. Se viceversa sta leggendo un file, nessun'altra può accedervi in scrittura. Se una applicazione sta usando una directory o leggendo un file, nessun'altra può cancellare o cambiare il nome al file o alla directory. Quindi bisogna fermare le altre applicazioni che stanno usando il file o la directory e ritentare.

203: object already exists (oggetto già esistente)
Causa probabile:
Il nome dell'oggetto specificato è quello di un oggetto che esiste già.
Suggerimenti per l'eliminazione:
Bisogna cancellare la directory o il file, se si vuole realmente usare quel nome.

204: directory not found (directory non trovata)

205: object not found (oggetto non trovato)
Causa probabile:
L'AmigaDOS non riesce a trovare il file o il device che è stato indicato: probabilmente si tratta di un errore di battitura o di suddivisione.

Suggerimenti per l'eliminazione:
Controllare i nomi dei device e dei file. Questo messaggio d'errore viene generato anche se si cerca di creare un file in una directory che non esiste.

206: invalid window (finestra non valida)
Causa probabile:
Sono state fornite dimensioni troppo ampie o troppo ridotte, oppure si è sbagliata la definizione di un'intera finestra (per esempio dimenticando la barra finale). Si può ricevere questo errore da NEWCLI se si indica un nome di device che non è una finestra.
Suggerimenti per l'eliminazione:
Bisogna specificare nuovamente la finestra.

210: invalid stream component name (nome del canale non valido)
Causa probabile:
Si è incluso un carattere non valido in un nome di file oppure tale nome è troppo lungo. Ogni file o directory deve avere un nome con un massimo di 30 caratteri; inoltre un filename non può contenere caratteri di controllo.

211: object not of required type (oggetto non del tipo richiesto)
Causa probabile:
Probabilmente avete tentato di effettuare un'operazione che richiede un nome di file e invece avete fornito un nome di directory o viceversa. Per esempio, potete aver impartito il comando TYPE dir, dove "dir" è una directory; infatti l'AmigaDOS non permette di usare il comando TYPE con le directory, ma solo con i file.
Suggerimenti per l'eliminazione:
Controllare l'uso dei comandi nel capitolo 2 di questo manuale.

213: disk not validated (disco non convalidato)
Causa probabile:
Si è appena inserito un disco e il processo di convalida è ancora in corso, oppure può trattarsi di un disco difettoso.
Suggerimenti per l'eliminazione:
Aspettare che il processo di convalida termini: normalmente impiega meno di un minuto. Se l'AmigaDOS non può convalidare il disco perché è difettoso, il disco rimane non convalidato. In questo caso si può solamente leggere dal disco e bisogna copiare le proprie informazioni in un altro disco.

214: disk write-protected (disco protetto in scrittura)
Causa probabile:
Il disco è protetto in scrittura. L'Amiga non può accedervi in scrittura, e

quindi non può neanche sovrascrivere le informazioni che sono già sul disco, ma solo leggerle, e non può memorizzare alcun tipo di dato.
Suggerimenti per l'eliminazione:
Salvare le proprie informazioni in un disco che non è protetto in scrittura oppure cambiare la posizione della linguetta di protezione sul disco.

215: rename across devices attempted (tentativo di effettuare il rename attraverso due device)
Causa probabile:
RENAME cambia il nome di un file solo in uno stesso device, sebbene possa essere usato per spostare un file da una directory a un'altra.
Suggerimenti per l'eliminazione:
Copiare il file nel device di destinazione e cancellarlo in quello sorgente.

216: directory not empty (directory non vuota)
Causa probabile:
Non si può cancellare una directory finché non è vuota.
Suggerimenti per l'eliminazione:
Cancellare il contenuto della directory, studiando la spiegazione del comando DELETE nel capitolo 2 di questo manuale.

218: device not mounted (device non configurato)
Causa probabile:
La parola "configurato" qui significa "inserito nel drive"; o si è commesso un errore di battitura oppure il disco che ha il nome desiderato non è configurato.
Suggerimenti per l'eliminazione:
Controllare la battitura o inserire il disco corretto.

220: comment too big (commento troppo lungo)
Causa probabile:
La nota apposta al file è più lunga del numero massimo di caratteri permesso (80).
Suggerimenti per l'eliminazione:
Riscrivere il commento rispettando il limite.

221: disk full (disco pieno)
Causa probabile:
Non c'è abbastanza spazio sul disco per effettuare l'operazione richiesta.
Suggerimenti per l'eliminazione:
Usare un altro disco oppure cancellare file o directory non necessari.

222: file is protected from deletion (il file è protetto dalla cancellazione)
Causa probabile:
Il file o la directory sono stati protetti dalla cancellazione.
Suggerimenti per l'eliminazione:
O si è cercato di cancellare questo file per sbaglio oppure lo si voleva cancellare veramente. In quest'ultimo caso, bisogna usare il comando PROTECT (descritto nel capitolo 2 di questo stesso manuale) per alterare lo stato della protezione. Si può usare anche il comando LIST per verificare quali protezioni sono attive per questo file particolare.

223: file is protected from writing (il file è protetto in scrittura)
Causa probabile:
Il file o la directory sono stati protetti dalla sovrascrittura.
Suggerimenti per l'eliminazione:
O si è cercato di scrivere in questo file per sbaglio oppure si voleva farlo veramente. In quest'ultimo caso, bisogna usare il comando PROTECT (descritto nel capitolo 2 di questo stesso manuale) per alterare lo stato della protezione. Si può usare anche il comando LIST per verificare quali protezioni sono attive per questo file particolare.

224: file is protected from reading (il file è protetto in lettura)
Causa probabile:
Il file o la directory sono stati protetti in lettura.
Suggerimenti per l'eliminazione:
O si è cercato di leggere questo file per sbaglio oppure lo si voleva leggere veramente. In quest'ultimo caso, bisogna usare il comando PROTECT (descritto nel capitolo 2 di questo stesso manuale) per alterare lo stato della protezione. Si può usare anche il comando LIST per verificare quali protezioni sono attive per questo file particolare.

225: not a DOS disk (non è un disco DOS)
Causa probabile:
Il disco nel drive non è un disco formattato con il DOS.
Suggerimenti per l'eliminazione:
Inserire nel drive un disco formattato correttamente oppure formattare il disco con il comando FORMAT.

226: no disk in drive (nessun disco nel drive)
Causa probabile:
Si è cercato di scrivere o leggere da un drive nel quale non è inserito alcun disco.
Suggerimenti per l'eliminazione:
Inserire un disco formattato con il DOS nel drive.

Errori del programmatore

209: packet request type unknown (tipo di packet non conosciuto)
Causa probabile:
Si è richiesto a un gestore di device di effettuare un'operazione che non può eseguire (per esempio il gestore della console non può cambiare nome a nulla).
Suggerimenti per l'eliminazione:
Controllare il codice della richiesta passata al gestore del device.

211: invalid object lock (lock dell'oggetto non valido)
Causa probabile:
Il lock utilizzato per l'oggetto non è valido.
Suggerimenti per l'eliminazione:
Controllare il proprio codice in modo che trasmetta alle funzioni AmigaDOS soltanto i lock corretti.

219: seek error (errore di seek)
Causa probabile:
Si è cercato di chiamare SEEK con argomenti non validi.
Suggerimenti per l'eliminazione:
Bisogna essere sicuri di effetture un SEEK solamente all'interno del file; non lo si può fare all'esterno dei suoi confini.

232: no more entries in directory (non ci sono più voci nella directory)
Causa probabile:
Non ci sono più voci nella directory che si sta esaminando.
Suggerimenti per l'eliminazione:
Questo codice d'errore indica che la funzione AmigaDOS EXNEXT non trova nella directory altre voci da esaminare e da restituire. È sufficiente smettere di chiamare EXNEXT.

Glossario

Argomenti
Informazioni aggiuntive fornite ai comandi.
Coda di output
Area di memoria (buffer) che mantiene i dati prima di scriverli in un file.
Comandi di editing
I comandi ricevuti dalla tastiera che controllano una sessione di lavoro con un editor.

Comando
Un'istruzione impartita direttamente al computer.
Comando corrente di alterazione stringa
Un'istruzione che cambia la stringa corrente.
Combinazione con il control
Una combinazione del tasto CTRL con una lettera o un simbolo. Il tasto CTRL deve essere premuto mentre si digita la lettera o il simbolo. Appare nella documentazione, per esempio, nella forma CTRL-A.
Command Line Interface (CLI)
Processo che decodifica l'input dell'utente.
Cursore, posizione corrente
La posizione nella quale si trova il cursore.
Delimitatori
Caratteri usati all'inizio e alla fine di una stringa per delimitarla.
Directory
Una serie di file.
Directory corrente
Può essere la directory radice oppure l'ultima impostata con il comando CD.
Directory radice
Il livello più in alto del filing system. I file e le directory contenuti nella directory radice hanno i nomi preceduti dai due punti (:).
Drive corrente
Il disk drive che è inserito e dichiarato come corrente. Il default è SYS:.
File
Una collezione di dati in relazione tra loro.
File destinazione
Il file nel quale si sta scrivendo o si deve scrivere.
Filename o nome di file
Un nome associato a un file per funzioni di identificazione.
File sequenziale
Un file nel quale si può raggiungere un certo punto percorrendolo sequenzialmente dall'inizio alla fine, finché non si raggiunge il punto desiderato.
File sorgente
Un file dal quale si leggono i dati.
Gestore del terminale o della console
Un processo che gestisce l'input e l'output del terminale o console.
Keyword
Argomento di un comando che deve essere indicato esplicitamente.
Linea corrente
La linea che EDIT controlla in un determinato momento.

Memoria
Il luogo nel quale il computer memorizza i propri dati e le proprie istruzioni.

Modo esteso
In questa modalità i comandi appaiono nella linea di comando e non vengono eseguiti finché non la si termina.

Modo immediato
In questa modalità i comandi sono eseguiti immediatamente.

Multitasking
L'esecuzione di due o più processi in parallelo, cioè nello stesso momento.

Nome di device
Il nome unico dato a un device, per esempio df0: = disk drive 0.

Nome di volume
Il nome unico associato a un disco.

Porzione di linea
Parte di una linea sulla quale agiscono i comandi di EDIT.

Priorità
L'importanza di un processo rispetto agli altri.

Processo
Un'operazione richiesta dal sistema operativo o dall'utente.

Puntatore
Puntatore al margine sinistro di una porzione di linea di EDIT. Si usa per definire la parte di linea da alterare.

Qualificatori
Caratteri che dichiarano condizioni aggiuntive riguardo a una stringa.

Sintassi
Il formato o la "grammatica" che si usa per un certo comando.

Spazi di trailing
Spazi che completano una riga prima del carattere CR (Carriage Return, ritorno carrello).

Stringa di caratteri
Sequenza di caratteri stampabili.

Stringa qualificata
Una stringa preceduta da uno o più qualificatori.

Task
Sinonimo di processo.

Template
Il metodo per definire la sintassi di ogni comando.

Wild card (jolly)
Simbolo usato per far combaciare un qualunque pattern.

Il Manuale dell'AmigaDOS per il programmatore

Indice

1. Programmazione dell'Amiga 199
2. Chiamate all'AmigaDOS 213
3. Il Macro Assembler. 233
4. Il linker 257

Appendice: input e output da console dell'Amiga. 269

Uso di Preferences

La dimensione di default del testo in un Amiga è di 60 caratteri per linea all'interno di una finestra CLI di larghezza massima. Numerosi programmatori preferiscono usare 80 caratteri per linea. È possibile modificare la dimensione del testo usando il programma Preferences presente nel disco Workbench; si deve però tener conto che la nuova larghezza del testo non agisce necessariamente sulle finestre precedentemente aperte. Per estendere le nuove dimensioni del testo a tutto il sistema occorre creare una nuova finestra, selezionare quella vecchia e cancellarla.

Seguite i passi seguenti:

1. Usate il comando NEWCLI.
2. Selezionate la vecchia finestra.
3. Usate il comando ENDCLI nella vecchia finestra per cancellarla.

Se modificate la selezione CLI il cambiamento può non avere un effetto immediato. Se invece salvate le nuove preferenze impostate tramite il programma Preferences ed eseguite un reboot del sistema, queste ultime saranno subito operative.

Capitolo 1

Programmazione dell'Amiga

Questo capitolo introduce il lettore alla programmazione in linguaggio C o Assembly in ambiente AmigaDOS.

1.1 Introduzione
1.2 Sviluppo di programmi per l'Amiga
1.2.1 Pronti per iniziare
1.2.2 Richiamo delle librerie residenti
1.2.3 Creazione di un programma eseguibile
1.3 Esecuzione di un programma in ambiente CLI
1.3.1 Ambiente iniziale in Assembly
1.3.2 Ambiente iniziale in C
1.3.3 Fallimento delle routine
1.3.4 Conclusione di un programma
1.4 Esecuzione di un programma in ambiente Workbench
1.5 Cross development
1.5.1 Cross development con un Sun Microsystem
1.5.2 Cross development in ambiente MS-DOS
1.5.3 Cross development con altri computer

1.1 Introduzione

L'ambiente di programmazione dell'AmigaDOS è disponibile nei sistemi Amiga, Sun e PC IBM.

Questo manuale assume che abbiate familiarità con il C o con l'Assembly. Non è tra i suoi scopi insegnare questi linguaggi.

1.2 Sviluppo di programmi per l'Amiga

Questa sezione descrive come sviluppare programmi per l'Amiga. Spiega di che cosa avete bisogno prima di iniziare a programmare, come richiamare le routine di sistema e come creare un file eseguibile sull'Amiga.

> **ATTENZIONE:** prima di fare QUALUNQUE cosa dovreste fare una copia di sicurezza del vostro disco sistema. Per le istruzioni riferitevi alla sezione 1.6 "Comandi d'uso più comune", nel *Manuale dell'Amiga-DOS per l'utente* in questo stesso volume.

1.2.1 Pronti per iniziare

Prima di iniziare a scrivere programmi per l'Amiga dovete disporre dei seguenti strumenti:

1. La documentazione dell'AmigaDOS e delle altre routine di sistema che potete richiamare. Per esempio vi serviranno i volumi *Programmare l'Amiga Vol. I e II*, pubblicati dalla *IHT Gruppo Editoriale*.
2. La documentazione riguardante il linguaggio che intendete usare. Se desiderate adottare l'Assemby o il C, questo manuale vi indicherà come impiegare questi strumenti, pur non contenendo nessuna delle informazioni specifiche che normalmente si possono trovare in un manuale d'uso dedicato interamente a questi linguaggi.
3. I file header che contengono le necessarie definizioni delle strutture dati dell'Amiga e i parametri per richiamare le routine di sistema necessarie. La Commodore-Amiga fornisce questi file header come file da includere sia per il C (di solito terminano in .h) sia per l'Assemby (terminano in .i). Per usare una particolare libreria residente devono essere inclusi uno o più file header contenenti le definizioni pertinenti. Ad esempio, per interagire con l'AmigaDOS da C si deve includere il file "dos.h".
4. Un assemblatore o un compilatore, eseguibile su un Amiga o in un ambiente cross development.
5. Il linker dell'Amiga, anch'esso disponibile su un Amiga o su un altro computer, così come le librerie standard dell'Amiga che contengono le funzioni, le routine di interfacciamento e diversi valori assoluti.
6. Strumenti per trasferire i programmi sull'Amiga se state usando un ambiente cross development.

1.2.2 Richiamo delle librerie residenti

Occorre notare che esistono due modi per richiamare le routine di sistema da un programma Assembly creato dall'utente. I programmatori in C richiamano semplicemente le funzioni come sono indicate. Di solito si richiama una routine di sistema in Assembly, memorizzando nel registro A6 il puntatore alla base della libreria (library base pointer) della determinata libreria residente, per poi saltare a un appropriato offset negativo da tale puntatore. Gli offset sono disponibili come valori esterni assoluti, nella libreria dell'Amiga, e sono indicati con nomi di forma _LVOname. Così, per esempio, una chiamata a una routine della libreria potrebbe essere JSR _LVOname(A6), dove A6 è stato caricato con l'appropriato puntatore alla base della libreria. Questi tipi di puntatori vengono resi disponibili dopo il richiamo della funzione OpenLibrary della libreria Exec; il puntatore alla base dell'Exec è contenuto nella locazione 4 (l'unica locazione assoluta usata dall'Amiga). Questa locazione è anche conosciuta come AbsExecBase ed è definita nel file Amiga.lib (riferirsi al volume *Programmare l'Amiga Vol. I* pubblicato dalla *IHT Gruppo Editoriale* per ulteriori informazioni riguardanti la libreria Exec).

Si possono richiamare alcune librerie residenti in RAM e la libreria dell'AmigaDOS in questo stesso modo, tenendo presente che la libreria dell'AmigaDOS si chiama "dos.library". Comunque non è necessario usare A6 per contenere il puntatore alla base di una libreria; si può usare qualunque altro registro se ne avete bisogno. In aggiunta si può richiamare l'AmigaDOS usando la possibilità di chiamata delle librerie residenti offerta dal linker. In questo caso si usa un JSR al punto di entrata e il linker annota il fatto che avete usato un riferimento a una libreria residente. Quando il vostro codice viene caricato in memoria, il Loader apre automaticamente la libreria e la richiude in vostra vece quando il programma viene rimosso. Il Loader aggiusta automaticamente i riferimenti al punto di entrata dell'AmigaDOS, in modo che si riferiscano all'offset corretto rispetto al puntatore alla base della libreria.

1.2.3 Creazione di un programma eseguibile

Per produrre un file eseguibile sull'Amiga si devono seguire i quattro punti illustrati di seguito. Ogni passo può essere compiuto usando l'Amiga oppure un opportuno computer per il cross development.

1. Prendete il listato sorgente del programma, che può essere stato scritto direttamente con un editor oppure trasferito da un altro computer. Il trasferimento del file sorgente può avvenire attraverso uno dei programmi di comunicazione disponibili in commercio, o adottando, se presenti nel proprio disco sistema, i programmi READ e DOWNLOAD.
2. Assemblate, o compilate, il programma.

3. Servitevi del linker per rendere il vostro programma eseguibile, in modo da includere qualunque routine di startup di cui abbiate bisogno e scandire la libreria dell'Amiga, o qualunque altra sia necessaria, per soddisfare tutti i riferimenti esterni.
4. Caricate il programma nell'Amiga e osservatelo mentre viene eseguito.

1.3 Esecuzione di un programma in ambiente CLI

Vi sono due modi per eseguire un programma: tramite il CLI o in ambiente Workbench. Questa sezione descrive il primo dei due.

L'esecuzione di un programma da CLI è un po' come usare una telescrivente (TTY) di vecchio tipo, orientata al trattamento di linee di testo. Nonostante questo, il CLI può risultare utile per molteplici compiti: ad esempio, per trasferire un proprio programma sull'Amiga come primo passo dello sviluppo. Per caricare ed eseguire il proprio programma si deve semplicemente digitare il nome del file che contiene il codice binario seguito da un certo numero di eventuali argomenti.

1.3.1 Ambiente iniziale in Assembly

Quando si carica un programma da CLI, si deve digitare il nome del programma e un insieme di argomenti. Si può inoltre specificare la ridirezione dell'input o dell'output facendo uso dei simboli “>” e “<”. Il CLI fornisce automaticamente tutte queste informazioni al programma quando quest'ultimo viene eseguito.

Il CLI, prima di mandare in esecuzione un programma, gli alloca uno stack di dimensioni iniziali pari a 4000 byte. Le dimensioni dello stack possono essere modificate per mezzo del comando STACK. L'AmigaDOS ricava lo stack dalla memoria libera ancora disponibile, prima che il programma venga eseguito; non è comunque lo stesso stack usato dal CLI. L'AmigaDOS pone nello stack un certo indirizzo di ritorno, che serve successivamente al CLI per riprendere il controllo e rimuovere il programma. Inoltre nello stack si trova, alla locazione 4(SP), la dimensione in byte dello stack stesso, che può risultare utile nel caso si desideri effettuare un controllo.

Quando il programma ha inizio, il registro A0 punta agli argomenti che sono stati indicati sulla linea di comando. L'AmigaDOS memorizza la linea degli argomenti nella memoria appartenente allo stack del CLI e il puntatore A0 rimane valido in tutto il programma. Il registro D0 indica il numero di caratteri presenti nella linea degli argomenti. Questi valori iniziali possono essere usati per leggere gli argomenti in modo da conoscere le richieste dell'utente. Occorre notare che tutti i registri possono essere distrutti, in senso software, da un programma utente.

Per avviare l'iniziale gestione dei file di input e di output, occorre richiamare le routine Input() e Output() dell'AmigaDOS, ricordandosi di

aprire la libreria dell'AmigaDOS prima di procedere. Le chiamate restituiscono gli handle dei file che si riferiscono allo standard input e output richiesto dall'utente. Lo standard input e output è normalmente il terminale, a meno che l'I/O non sia stato ridiretto includendo ">" o "<" nella linea degli argomenti. Questi file handle non devono venir chiusi dal programma; il CLI li apre e li chiude in vostra vece.

1.3.2 Ambiente iniziale in C

Quando si programma in C si deve sempre includere il codice di startup come primo elemento dell'input del linker. Questo significa che il linker fa iniziare il vostro programma al punto di entrata della routine di startup. Questa provvede a scandire la lista degli argomenti e rende disponibili gli argomenti stessi tramite "argv", e il loro numero per mezzo di "argc", come di consueto. Inoltre apre la libreria dell'AmigaDOS e richiama automaticamente le funzioni Input( ) e Output( ) ponendo gli handle risultanti in "stdin" e "stdout". Infine esegue una chiamata alla funzione C "main".

1.3.3 Fallimento delle routine

Le funzioni dell'AmigaDOS restituiscono per la maggior parte uno zero se falliscono; le eccezioni sono le funzioni Read e Write che restituiscono -1 quando incontrano un errore. Se si riceve di ritorno un errore si può richiamare IoErr( ) per ottenere maggiori informazioni sui motivi del fallimento. IoErr( ) restituisce un numero intero che corrisponde al codice di errore completo, sulla base del quale si possono intraprendere diverse operazioni a seconda della ragione esatta per cui la routine è fallita. Una lista completa dei codici d'errore e dei messaggi è disponibile alla fine del *Manuale dell'AmigaDOS per l'utente* in questo stesso libro.

1.3.4 Conclusione di un programma

Per uscire da un programma è sufficiente dare un semplice RTS usando lo stack pointer iniziale (SP). In questo caso si deve fornire un codice di ritorno memorizzandolo nel registro D0. Se il programma ha funzionato il codice è zero, altrimenti è un valore positivo. Se il codice di ritorno è diverso da zero il CLI lo interpreta come segnalazione di un errore. A seconda del livello di fallimento corrente, impostato facendo uso del comando FAILAT, un processo CLI non interattivo, come quello che esegue una sequenza di comandi indicata per mezzo di un comando EXECUTE, può arrestarsi o continuare. Un programma scritto in C può terminare effettuando un return da "main", restituendo di conseguenza il controllo alla routine di startup che

provvede ad azzerare il contenuto di D0 ed effettuare un RTS.

In alternativa, un programma può richiamare la funzione Exit dell'Amiga-DOS, la quale richiede come argomento il codice di ritorno. In questo modo si può uscire dal programma in qualunque momento senza preoccuparsi di quale sia il valore dello stack pointer.

Ora diventa importante porre l'accento sul fatto che l'AmigaDOS non controlla automaticamente le risorse messe a disposizione del programma: la gestione è infatti demandata totalmente al programmatore. Qualunque file aperto da un programma utente deve essere chiuso prima che il programma termini. Analogamente, qualunque lock venga ottenuto deve essere rilasciato, qualunque parte di programma venga caricata deve essere rimossa e tutta la memoria allocata deve essere resa nuovamente disponibile. Naturalmente si possono verificare alcuni casi in cui non si desidera rilasciare tutte le risorse, per esempio quando si è scritto un programma che carica in memoria un segmento di codice per poterlo usare successivamente. Questo è perfettamente accettabile, ma si deve disporre di un meccanismo per rilasciare eventualmente la memoria allocata, i lock dei file e così via. Nella sezione 3.7 del capitolo 3 di questo manuale vengono spiegate le strutture lock dei file. Consultatelo per ottenere maggiori dettagli.

1.4 Esecuzione di un programma in ambiente Workbench

Per eseguire un programma in ambiente Workbench bisogna saper distinguere i diversi modi in cui un programma può essere eseguito con l'Amiga. In ambiente CLI il programma viene eseguito come parte del processo CLI da cui eredita gli stream di I/O, altre informazioni, e gli argomenti forniti.

Quando un programma viene mandato in esecuzione da Workbench, l'AmigaDOS attiva un processo la cui esecuzione avviene contemporaneamente a quella del Workbench stesso. Il Workbench carica il programma in memoria e poi genera un messaggio per fare in modo che venga eseguito. Si deve quindi attendere questo messaggio iniziale prima di procedere a qualunque operazione; inoltre il messaggio deve essere memorizzato perché l'applicazione possa restituirlo al Workbench quando il programma è terminato, in modo che il Workbench stesso rimuova tutto il codice del programma.

Per i programmatori in C tutto questo viene svolto adottando una differente routine di startup. I programmatori in Assembly devono invece fare tutto da soli.

Bisogna inoltre tener presente che un programma eseguito come nuovo processo iniziato dal Workbench, non possiede canali di input e di output di default. Bisogna quindi assicurarsi che il programma apra tutti i canali di I/O di cui necessita, e che li richiuda quando ha terminato.

1.5 Cross development

Se state usando un ambiente cross development, avete bisogno di caricare il vostro programma in un Amiga. Questa sezione descrive il supporto speciale che la Commodore-Amiga fornisce agli ambienti MS-DOS e Sun Microsystem. Descrive inoltre come effettuare il cross development in altri ambienti senza questo supporto speciale.

1.5.1 Cross development con un Sun Microsystem

Gli strumenti disponibili in un Sun Microsystem per il cross development, includono un assemblatore, un linker e due compilatori C. Il formato per gli argomenti da fornire all'assemblatore e al linker è identico a quello usato per un Amiga in ambiente CLI. Il compilatore C della Greenhills è disponibile solo nel Sun Microsystem ed è descritto qui di seguito.

Questo compilatore è chiamato "metacc" e accetta diversi tipi di file, assumendo che i filename terminanti in .c indichino i programmi sorgente in C. Il compilatore compila i file .c e memorizza il codice oggetto risultante nella directory corrente, mantenendo lo stesso filename ma modificando il suffisso .c in .obj, che indica appunto un file oggetto. Il compilatore assume che i file terminanti in .asm siano programmi sorgente in assembly. Si può usare l'assemblatore per assemblare tali programmi e creare nella directory corrente i corrispondenti file oggetto terminanti in .obj.

Il compilatore "metacc" è in grado di ricevere numerosi argomenti nel seguente formato:

metacc [<opt1>[,<opt2>[,...<optn>]]][<file>[,...<filen>]]

Le opzioni disponibili sono le seguenti:

-c -g -go -w -p -pg -O[<optflag>] -fsingle
-S -E -C -X70 -o<output> -D<nome=def>
-U<nome> -I<dir> -B<stringa> -t[p012]

e comunicano a "metacc" di compiere le seguenti operazioni:

-c compila solamente il programma, sopprimendo la fase di caricamento della compilazione e forzando la produzione di un file oggetto anche se compila un solo programma.

-g produce una tavola addizionale dei simboli per l'uso del debugger dbx e per passare il flag -lg a ld.

-go produce una tavola dei simboli addizionale, in un vecchio formato usato dal debugger adb e passa il flag -lg a ld.

-w sopprime tutti i messaggi di avvertimento (warning).

-p produce il codice per un profiler il quale conta il numero di volte che una routine viene richiamata. Se avviene un caricamento rimpiazza la routine di startup standard con una che viene richiamata automaticamente dal monitor, e fa uso di una speciale libreria al posto della libreria C standard.
Per generare un resoconto dell'esecuzione è necessario usare il programma prof.

-pg produce un codice analogo a -p, ma invoca al momento dell'esecuzione un meccanismo di registrazione che genera statistiche più approfondite e produce il file gmon.out al termine.
Per generare un resoconto dell'esecuzione si deve usare il programma gprof.

-O[<optflag>] usa l'ottimizzatore del codice oggetto per migliorare il codice generato.
<optflag>, se indicato, viene trasferito nella linea di comando dell'ottimizzatore: in pratica -O viene usato per passare i flag delle opzioni all'ottimizzatore.

-fsingle usa l'aritmetica in precisione singola nei calcoli che riguardano numeri in virgola mobile (dichiarati float); quindi non converte tutto in double (doppia precisione) come farebbe per default.
Nota: *i parametri in virgola mobile vengono convertiti ugualmente in doppia precisione, e i valori restituiti dalle funzioni sono anch'essi in doppia precisione.*

> **ATTENZIONE:** alcuni programmi vengono eseguiti più velocemente usando l'opzione -fsingle, ma occorre fare attenzione al fatto che si può perdere qualcosa a causa della minore precisione dei valori intermedi.

-S compila in C i programmi indicati e lascia l'output in linguaggio Assembly nei file corrispondenti che terminano in .obj.

-E — esegue solamente il pre-processore C sui programmi in C specificati, mandando il risultato verso l'output standard.

-C — evita che il pre-processore C rimuova i commenti.

-X70 — genera il codice usando il formato in virgola mobile dell'Amiga. Questo codice è compatibile con la libreria matematica in virgola mobile del ROM Kernel fornita dall'Amiga.

-o<output> — chiama il file finale di output "output". Se si usa questa opzione il file a.out resta invariato.

-D<nome=def> — definisce un "nome" per il pre-processore C come se si fosse usato #define. Se non viene fornita nessuna definizione, per default gli viene assegnato il valore "1".

-U<nome> — rimuove qualunque definizione iniziale relativa a "nome".

-I<dir> — ricerca sempre i file #include il cui nome non inizia con "/", prima nella directory dell'argomento "file", poi nella directory "dir" specificata con l'opzione -I, e infine nella directory /usr/include.

-B<stringa> — ricerca i passaggi di sostituzione del compilatore nei file specificati da <stringa> con i suffissi finali cpp, ccom e c2. Se "stringa" è vuota usa una versione di backup.

-t[p012] — ricerca solo i passaggi del compilatore indicati nei file i cui nomi sono ricavati dall'opzione -B. In assenza dell'opzione -B assume per default che "stringa" sia /usr/new/.
Le combinazioni di numeri e lettere che si possono indicare con l'opzione -t hanno i seguenti significati:

- p cpp - il preprocessore C
- 0 metacom - entrambe le fasi del compilatore C, ma non l'ottimizzatore
- 1 ignorato in questo sistema - questa opzione si riferirebbe alla seconda fase di un compilatore a due fasi, ma nel sistema Sun ccom include entrambe le fasi.
- 2 c2 - l'ottimizzatore del codice oggetto.

Il compilatore metacc assume che gli altri argomenti siano programmi oggetto, argomenti di opzioni caricate oppure librerie di programmi oggetto. A meno che non venga indicato -c, -S oppure -E, metacc carica questi programmi e librerie insieme ai risultati di qualunque compilazione o assemblaggio specificato (nell'ordine dato), per produrre un programma eseguibile chiamato a.out. Per chiamare il file di output con un nome diverso da a.out si può usare l'opzione -o<nome>.

Se un singolo programma in C viene compilato e caricato tutto in una volta, viene cancellato il file .o intermedio.

La Tavola 1-A mostra una lista dei nomi dei file speciali relativi a metacc uniti a una breve descrizione.

File speciali

Descrizione dei file	**Filename**
File sorgente in C	file.c
File sorgente in Assembly	file.asm
File oggetto	file.o
Libreria di file oggetto	file.lib
File di output eseguibile	a.out
File temporaneo	/tmp/ctm
Pre-processore C	/lib/cpp
Compilatore	/lib/ccom
Ottimizzatore opzionale	/lib/c2
Startoff di runtime	/lib/crt0.o
Startoff per il profiler	/lib/mcrt0.o
Startoff per il profiler gprof	/usr/lib/grt0.o
Libreria standard	/lib/libc.a
Libreria del profiler	/usr/lib/libc_p.a
Directory standard #include	/usr/include
File prodotti da prof per essere analizzati	mon.out
File prodotti da gprof per essere analizzati	gmon.out

Tavola 1-A: nomi di file speciali relativi a metacc

I file prodotti con il linker in un Sun possono essere trasferiti in un Amiga in tre modi diversi: il primo e più semplice richiede un BillBoard, il secondo una porta parallela e il terzo una linea seriale.

Se si dispone della periferica speciale chiamata BillBoard, si possono trasferire i file prodotti con il linker, che per convenzione terminano con .ld, nella maniera seguente:

1. Mandate in esecuzione il programma "binload" sul Sun

```
binload -p &
```

(è necessario farlo una volta sola)

2. Quindi digitate sull'Amiga

```
download <filename Sun> <filename Amiga>
```

3. Per eseguire il programma digitate

```
<filename Amiga>
```

Per esempio:

digitate sul Sun

```
binload -p &
```

digitate sull'Amiga

```
download test.ld test
```

oppure

```
download /usr/commodore/amiga/V24/examples/DOS/test.ld test
```

e infine digitate sull'Amiga

```
test
```

DOWNLOAD accede ai file del Sun relativi alla directory dalla quale è stato mandato in esecuzione binload. Se la directory del Sun è /usr/commodore/ amiga/V24/examples/DOS, come nell'esempio precedente, è necessario specificare anche solo il nome del file (test.ld). Se non ci si ricorda la directory da cui è stato eseguito binload occorre indicare l'intero nome del file, completo della directory di appartenenza. Per arrestare binload bisogna impartire un "ps" seguito da un "kill" con il PID. Un reset software del computer comunica a binload di scrivere un messaggio nel proprio output standard (il default è rappresentato dalla finestra dal quale è stato eseguito). Se il trasferimento si interrompe occorre premere CTRL-C sull'Amiga per sopprimere il programma DOWNLOAD (per ulteriori informazioni sull'uso di CTRL-C, CTRL-D, CTRL-E e CTRL-F riferirsi alla sezione 3.2 del *Manuale dell'AmigaDOS per l'utente*).

Se non si possiede un BillBoard, si possono trasferire i file attraverso la

porta parallela. Per farlo, dovete compiere i passi seguenti:

1. Mandate attraverso la porta parallela il file ASCII da trasferire nell'Amiga, per mezzo del comando

   ```
   send demo.ld
   ```

   Se non viene fornito alcun argomento a "send", viene usato l'input standard. La periferica di output di default è /dev/lp0 che normalmente è corretta. Per cambiare l'output di default si può usare l'argomento -o.

2. Digitate sull'Amiga

   ```
   READ demo
   ```

   READ legge i caratteri provenienti dalla porta parallela e li pone nel file chiamato "demo".

3. Una volta che READ ha terminato si deve digitare

   ```
   demo
   ```

   per eseguire il programma demo.

   Si possono trasferire i file anche per via seriale. Per fare questo occorre compiere i seguenti passi:

1. Convertite il file binario caricabile in un file ASCII terminante con Q digitando

   ```
   convert <demo.ld >demo.dl
   ```

   (dove .dl significa per convenzione file da trasferire, dall'inglese DownLoad file). La regola precedente è presente nel makefile "makeamiga", fornito a corredo (riferirsi al capitolo 2 del *Manuale dell'AmigaDOS di riferimento tecnico* per ulteriori dettagli sui file binari gestiti dall'Amiga).

2. Digitate

   ```
   tip amiga
   ```

3. Sull'Amiga digitate

   ```
   READ demo serial
   ```

4. All'interno di tip scrivete

```
~>demo.dl
```

5. Quando READ ha completato la lettura si può digitare sull'Amiga "demo" per eseguire il programma.

ATTENZIONE: spesso la porta seriale del Sun interrompe i trasferimenti senza un motivo apparente. Se ciò accade si deve inizializzare nuovamente il sistema.

Se la porta seriale interrompe il trasferimento, si deve inizializzare nuovamente il Sun e in seguito digitare

```
tip
```

e, all'interno di tip, aggiungere

```
Q
```

affinché READ completi il suo lavoro sull'Amiga. Una volta che questo è stato fatto si deve iniziare un nuovo READ e si devono digitare sul Sun i seguenti simboli:

```
~>
```

1.5.2 Cross development in ambiente MS-DOS

Per opererare in cross development usando un computer che adotta lo standard MS-DOS, sono necessari diversi strumenti forniti nella directory \V25\bin inclusa nel relativo pacchetto di sviluppo. In essa risiedono il compilatore C, l'assemblatore, il linker e inoltre i comandi che vengono utilizzati nella fase di trasferimento. Per i comandi eseguiti da MS-DOS si usa la stessa sintassi adottata per il CLI.

Per trasferire un file attraverso la porta seriale di un PC IBM (chiamata AUX) si devono compiere i passi seguenti:

1. Digitate sull'Amiga

```
READ file SERIAL
```

2. Sul PC digitate

```
convert <file.ld >AUX:
```

3. Sull'Amiga ora si può scrivere

```
file
```

per eseguire il programma.

1.5.3 Cross development con altri computer

Per il cross development con altri computer è necessario disporre di un compilatore o di un assemblatore adatto, e di tutti i file da includere che definiscono le varie funzioni disponibili. Inoltre si deve poter eseguire il linker dell'Amiga ALINK sul proprio computer oppure sull'Amiga. Infine bisogna poter convertire un file binario in una serie di valori ASCII esadecimali terminanti con un Q (è in questo formato che READ accetta i dati) e bisogna trovare il modo di trasferire questi dati alla porta parallela o a quella seriale.

Una volta che si è creato un file binario, lo si deve trasferire nell'Amiga usando il comando READ, così come è descritto nella sezione 1.5.2 di questo manuale. Se si dispone del linker dell'Amiga sul proprio computer, si può trasferire il file binario completo eseguibile; altrimenti si devono trasferire i file oggetto binari nel formato accettato da ALINK, per poi eseguire la fase di link sull'Amiga.

Capitolo 2

Chiamate all'AmigaDOS

Questo capitolo descrive le funzioni fornite dalla libreria residente dell'AmigaDOS. Allo scopo di aiutare il programmatore si fornisce: una spiegazione della sintassi, una descrizione completa di ogni funzione e una tavola di consultazione rapida delle funzioni disponibili.

2.1 Sintassi
2.1.1 Valori dei registri
2.1.2 Maiuscolo e minuscolo
2.1.3 Valori booleani di ritorno
2.1.4 Valori
2.1.5 Formato, argomento e risultato
2.2 Le funzioni dell'AmigaDOS
2.3 Tavola di consultazione rapida

2.1 Sintassi

La sintassi usata in questo capitolo mostra la chiamata di ogni funzione dell'AmigaDOS secondo il formato del linguaggio C, e i registri corrispondenti da utilizzare quando si programma in Assembly.

2.1.1 Valori dei registri

Le combinazioni lettera/numero (D0...Dn) rappresentano i registri. Il testo che si trova alla sinistra del segno di uguale rappresenta il risultato di una funzione. Il registro (per esempio D0) che appare sotto tale testo è quello che conterrà il valore del risultato. Il testo che si trova alla destra di un segno di uguale rappresenta una funzione e i propri argomenti, dove la parte racchiusa nelle parentesi è la lista degli argomenti. Il registro (per esempio D2) che appare sotto un argomento è quello che dovrà contenere il valore numerico dell'argomento stesso.

Non tutte le funzioni restituiscono un risultato.

2.1.2 Maiuscolo e minuscolo

La differenza tra caratteri maiuscoli e minuscoli è significativa; quando un carattere è maiuscolo deve essere trascritto in maiuscolo, quando è minuscolo in minuscolo. Per esempio, il puntatore "FileInfoBlock" deve essere trascritto con la prima lettera di ogni parola in maiuscolo.

2.1.3 Valori booleani di ritorno

-1 (VERO oppure SUCCESSO), 0 (FALSO oppure FALLIMENTO).

2.1.4 Valori

Tutti i valori sono long word (cioè valori di 4 byte, ovvero 32 bit). I valori riferiti a stringhe sono puntatori da 32 bit che individuano una serie di caratteri terminati da un carattere NULL.

2.1.5 Formato, argomento e risultato

Riferirsi ad "Argomento:" e "Risultato:" per ulteriori informazioni riguardo alla sintassi usata dopo "Formato:". Risultato descrive quanto viene restituito dalla funzione (si trova alla sinistra del segno di uguale). Argomento descrive i parametri che la funzione si aspetta di trattare (la lista all'interno delle parentesi). La Tavola 2-A aiuta la comprensione della sintassi.

Formato della funzione	risultato = Funzione(argomento)	
	Registro	Registro
Esempio	lock = CreateDir(nome)	
	D0	D1

Tavola 2-A: formato delle Funzioni e dei Registri

2.2 Le funzioni dell'AmigaDOS

Questa sezione descrive le funzioni messe a disposizione dalla libreria residente dell'AmigaDOS. Le funzioni sono suddivise in tre gruppi d'azione: trattamento dei file, trattamento dei processi e caricamento del codice. All'interno di ciascun gruppo le funzioni sono ordinate alfabeticamente. Dopo ogni nome di funzione si trova una breve descrizione del suo scopo, una spiegazione del formato e dei registri coinvolti, una completa descrizione della funzione e la sintassi degli argomenti e del risultato. Per usare queste funzioni, occorre eseguire un link con amiga.lib. Le strutture dati citate nelle descrizioni delle funzioni che seguono, sono illustrate nel capitolo 3 del *Manuale dell'AmigaDOS di riferimento tecnico.*

Trattamento dei file

Close

Scopo: chiudere un file di input o output.

Formato: *Close(file)*
D1

Argomento: file - handle di un file

Descrizione: l'handle del file "file" indica quale file deve essere chiuso. L'handle di un file si ottiene come risultato di una chiamata a Open. Bisogna ricordarsi di chiudere esplicitamente tutti i file aperti in un programma, con l'accortezza però di non chiudere gli handle dei file ereditati da aperture avvenute in altri programmi.

CreateDir

Scopo: creare una nuova directory.

Formato: *lock = CreateDir(nome)*
D0 D1

Argomento: nome - puntatore alla stringa del nome

Risultato: lock - puntatore a un lock

Descrizione: CreateDir crea, se è possibile, una nuova directory che abbia il nome indicato e restituisce un errore se fallisce. Occorre ricordare che l'AmigaDOS può creare directory solo in quelle periferiche che lo permettono, per esempio i dischi.

Se viene restituito uno zero, significa che l'AmigaDOS ha incontrato un errore (per esempio un disco protetto in scrittura) e occorre quindi chiamare IoErr(); altrimenti il valore restituito è un puntatore a un lock di lettura condiviso sulla nuova directory.

CurrentDir

Scopo: rendere una directory associata a un certo lock la nuova directory corrente.

Formato: *lock_vecchio = CurrentDir(lock)*
D0 D1

Argomento: lock - puntatore a un lock

Risultato: lock_vecchio - puntatore a un lock

Descrizione: CurrentDir rende corrente una directory associata a un certo lock (vedere anche LOCK). Restituisce il lock della precedente directory corrente.

Un valore pari a zero, in questo caso, rappresenta un risultato valido e indica che la precedente directory corrente era la radice del disco di startup iniziale.

DeleteFile

Scopo: cancellare un file o una directory.

Formato: *successo = DeleteFile(nome)*
D0 D1

Argomento: nome - puntatore alla stringa del nome

Risultato: successo - valore booleano

Descrizione: DeleteFile cerca di cancellare il file o la directory "nome". Restituisce un errore se la cancellazione fallisce. Per cancellare una directory occorre prima aver cancellato tutti i file in essa presenti.

DupLock

Scopo: duplicare un lock.

Formato: *lock_nuovo = DupLock(lock)*
D0 D1

Argomento: lock - puntatore a un lock

Risultato: lock_nuovo - puntatore a un lock

Descrizione: Duplock prende un lock di lettura condiviso del filing system e restituisce un altro lock relativo allo stesso oggetto (file o directory). Non è possibile creare una copia di un lock di scrittura (per maggiori informazioni relative ai lock fare riferimento a LOCK).

Examine

Scopo: esaminare una directory o un file associato a un lock.

Formato: *successo = Examine(lock, FileInfoBlock)*
D0 D1 D2

Argomento: lock - puntatore a un lock
FileInfoBlock - puntatore a un blocco di informazioni per un file (file info block)

Risultato: successo - valore booleano

Descrizione: Examine pone nel FileInfoBlock le informazioni che riguardano il file o la directory associata al lock indicato. Queste informazioni includono il nome, la dimensione, la data di creazione e se si tratta di una directory o di un file.

Nota: *FileInfoBlock deve essere allineato in memoria secondo una longword. In linguaggio C, per assicurarsi che l'allineamento avvenga è possibile richiamare la funzione AllocMem (riferirsi a Programmare l'Amiga Vol. I per ulteriori dettagli sulla funzione dell'exec AllocMem).*

Examine restituisce un codice pari a zero se fallisce.

ExNext

Scopo: esaminare la voce successiva presente in una directory.

Formato: *successo = ExNext(lock, FileInfoBlock)*
D0 D1 D2

Argomento: lock - puntatore a un lock
FileInfoBlock - puntatore a un blocco di informazioni per un file

Risultato: successo - valore booleano

Descrizione: questa routine richiede, come argomenti, un lock (generalmente associato a una directory) e un FileInfoBlock aggiornato con una precedente chiamata a Examine. Il FileInfoBlock contiene le informazioni riguardanti il primo file o subdirectory presente nella directory associata al lock. ExNext modifica il contenuto del FileInfoBlock in modo che a ogni successiva chiamata della funzione siano presenti le informazioni riguardanti la voce seguente della directory.

ExNext restituisce un codice pari a zero se fallisce per una qualunque ragione. Il fallimento potrebbe ad esempio verificarsi se viene raggiunta l'ultima voce presente nella directory. Comunque IoErr() può fornire un codice di ritorno che identifica meglio la causa esatta del fallimento. Quando ExNext termina dopo l'ultima voce, IoErr() restituisce il codice ERROR_NO_MORE_ENTRIES, (errore, non ci sono altre voci).

Quindi, per esaminare il contenuto di una directory si devono compiere i passi seguenti:

1) Usare Examine per ottenere il FileInfoBlock della directory che si desidera esaminare.

2) Passare a ExNext il lock relativo alla directory e il FileInfoBlock ottenuto dalla precedente chiamata a Examine.

3) Continuare a chiamare ExNext finché non fallisce.
Se il codice contenuto in IoErr() è uguale a ERROR_NO_MORE_ENTRIES, vuol dire che avete finito.

4) Se la natura dei dati che si stanno ispezionando non è nota, è sufficiente controllare il campo del tipo presente nel FileInfoBlock restituito da Examine, per sapere se si tratta di un file oppure di una directory: una chiamata a ExNext è valida solo se il FileInfoBlock iniziale appartiene a una directory.

Il campo del tipo, presente nel FileInfoBlock, può contenere due valori: se è negativo l'oggetto esaminato è un file, se è positivo è invece una directory.

Info

Scopo: fornire informazioni riguardo a un disco.

Formato: *successo = Info(lock, Info_Data)*
D0 D1 D2

Argomento: lock - puntatore a un lock
Info_Data - puntatore a una struttura Info_Data

Risultato: successo - valore booleano

Descrizione: Info fornisce le informazioni riguardanti qualunque disco in uso. "Lock" si riferisce al disco o a qualunque file presente sul disco. Info restituisce una struttura Info_Data contenente le informazioni riguardo alla dimensione del disco, al numero dei blocchi liberi e a qualunque errore software. La struttura Info_Data deve essere allineata secondo una longword.

Input

Scopo: ricevere l'handle del file per il device di input di default.

Formato: *file = Input()*
D0

Risultato: file - handle di un file

Descrizione: serve per identificare l'handle del file iniziale di input del programma (per identificare l'output iniziale si veda OUTPUT).

IoErr

Scopo: ricevere ulteriori informazioni dal sistema.

Formato: *errore = IoErr()*
D0

Risultato: errore - numero intero

Descrizione: le routine di I/O restituiscono uno zero per indicare un errore. Quando si verifica un errore si può chiamare questa funzione per ottenere maggiori informazioni. Alcune routine fanno uso di IoErr, per esempio DeviceProc, per fornire un risultato secondario.

IsInteractive

Scopo: scoprire se un file è connesso a un terminale virtuale.

Formato: *status = IsInteractive(file)*
D0 D1

Argomento: file - handle di un file

Risultato: status - valore booleano

Descrizione: la funzione IsInteractive restituisce un valore booleano che indica se il file associato all'handle indicato è connesso a un terminale virtuale oppure no.

Lock

Scopo: ottenere un lock di una directory o di un file.

Formato: *lock = Lock(nome, modo_accesso)*
D0 D1 D2

Argomento: nome - puntatore alla stringa
modo_accesso - numero intero

Risultato: lock - puntatore a un lock

Descrizione: Lock restituisce, se è possibile, un lock del file o della directory "nome". Se il modo di accesso è ACCESS_READ il lock che si ottiene è un lock di lettura condiviso; se il modo d'accesso è invece ACCESS_WRITE si ottiene un lock esclusivo di scrittura. Questo perché essendo il sistema multitasking, può capitare che diversi task accedano al file in lettura simultaneamente, ma non più di uno può avere accesso in scrittura nello stesso momento. Se Lock fallisce, cioè se non riesce a ottenere un lock del file o della directory, restituisce uno zero.

Il tempo necessario per una chiamata a Lock è minore di quello necessario per una chiamata a Open e quindi, se si desidera accertare l'esistenza di un file, si può usare Lock. Naturalmente una volta verificato che il file esiste, si deve usare Open per aprirlo.

Open

Scopo: aprire un file di input o di output.

Formato: *file = Open(nome, modo_accesso)*
D0 D1 D2

Argomento: nome - puntatore alla stringa
modo_accesso - numero intero

Risultato: file - handle di un file

Descrizione: Open apre il file "nome" e restituisce l'handle di quel file. Se il modo d'accesso è MODE_OLDFILE (=1005), viene aperto un file già esistente, per la lettura o la scrittura. Comunque Open crea un nuovo file per la scrittura se il modo d'accesso scelto è MODE_NEWFILE (=1006). Il "nome" può essere un filename (eventualmente preceduto da un nome di device), un device semplice come NIL:, una indicazione di finestra come CON: o RAW: seguita dagli eventuali parametri, oppure un * rappresentante la finestra corrente. Esiste, a partire dalla versione 1.2, un nuovo modo d'accesso per aprire i file: MODE_READWRITE (=1004). Questo modo apre un file esistente con un lock esclusivo. In. aggiunta si può usare l'identificatore MODE_READONLY come sinonimo di MODE_OLDFILE.

Per ulteriori dettagli sui device NIL:, CON: e RAW: vedere il capitolo 1 del *Manuale dell'AmigaDOS per l'utente* in questo stesso libro. Se Open non riesce ad aprire il file "nome" per un qualunque motivo, restituisce il valore zero. In questo caso si può effettuare una chiamata alla routine IoErr() per ottenere il codice d'errore secondario.

Per verificare se un file esiste si può usare LOCK.

Output

Scopo: ricevere l'handle del file per il device di output di default.

Formato: *file = Output()*
D0

Risultato: file - handle di un file

Descrizione: serve per identificare il file di output iniziale di un programma (per identificare l'input iniziale si usa INPUT).

ParentDir

Scopo: ottenere la directory genitore di un file o di un'altra directory.

Formato: *lock2 = ParentDir(lock1)*
D0 D1

Argomento: lock1 - puntatore a un lock

Risultato: lock2 - puntatore a un lock

Descrizione: questa funzione restituisce un lock associato alla directory genitore di un file o di un'altra directory. In pratica ParentDir riceve il lock associato a un file o a un'altra directory e restituisce quello associato alla loro directory precedente.

Nota: *il risultato di ParentDir può essere uno zero se si tratta della directory radice del filing system corrente.*

Read

Scopo: leggere byte di dati da un file.

Formato: *lunghezza_attuale = Read(file, buffer, lunghezza)*
D0 D1 D2 D3

Argomento: file - handle di un file
buffer - puntatore a un buffer
lunghezza - numero intero

Risultato: lunghezza_attuale - numero intero

Descrizione: si possono copiare dati usando una combinazione di Read e Write. Read legge byte di informazioni da un file aperto, rappresentato dall'argomento "file", e li memorizza nel buffer di memoria indicato. Read cerca di leggere tanti byte quanti ne stanno nel buffer come indicato dal valore di lunghezza. Bisogna sempre essere sicuri che il valore indicato come lunghezza rappresenti effettivamente la dimensione del buffer. Read può restituire un valore indicante che sono stati letti meno byte di quanti ne fossero richiesti, per esempio quando si legge una linea di dati digitati alla tastiera.

Il valore restituito è la lunghezza delle informazioni effettivamente lette. Quando "lunghezza_attuale" è maggio-

re di zero significa che il valore espresso è uguale al numero dei caratteri letti. Un valore pari a zero significa che si è raggiunta la fine del file, mentre un errore viene indicato dal valore -1. Quando Read opera con la console, restituisce un valore solo se viene incontrato il carattere CR o se il buffer viene riempito interamente.

Una chiamata a Read inoltre modifica il valore di IoErr(). La funzione IoErr fornisce maggiori informazioni riguardo a un errore (lunghezza_attuale è uguale a -1) quando viene richiamata.

Rename

Scopo: cambiare nome a una directory o un file.

Formato: *successo = Rename(nome_vecchio, nome_nuovo)*
D0 D1 D2

Argomento: nome_vecchio - puntatore alla stringa
nome_nuovo - puntatore alla stringa

Risultato: successo - valore booleano

Descrizione: Rename cerca di cambiare il nome della directory o del file specificato come "nome_vecchio" con il nome "nome_nuovo". Se il file o la directory "nome_nuovo" esiste già, Rename fallisce e restituisce un errore.

Sia "nome_vecchio" che "nome_nuovo" possono essere filename complessi che contengono anche indicazioni di directory. In questo caso il file viene mosso da una directory a un'altra purché la seconda esista già e non debba essere creata.

Nota: *non è possibile cambiare nome a un file da un volume a un altro.*

Seek

Scopo: muoversi a una posizione logica in un file.

Formato: *posizione_vecchia = Seek(file, posizione, modo)*
D0 D1 D2 D3

Argomento: file - handle di un file
posizione - numero intero
modo - numero intero

Risultato: posizione_vecchia

Descrizione: Seek imposta la posizione del puntatore di lettura/scrittura, relativo al file "file", alla posizione "posizione". Sia Read che Write si riferiscono a questa posizione come punto di partenza per scrivere o leggere. Se tutto funziona a dovere il valore restituito individua la precedente posizione nel file. Se si verifica un errore il valore restituito è -1. È possibile usare IoErr() per ottenere maggiori informazioni riguardo all'errore che si è verificato.

"Modo" può essere OFFSET_BEGINNING (=1) (offset a partire dall'inizio), OFFSET_CURRENT (=0) (offset a partire dalla posizione corrente) oppure OFFSET_END (=−1) (offset a partire dal termine). "Modo" viene usato per indicare la posizione di partenza relativa. Per esempio 20 con OFFSET_CURRENT significa 20 byte in avanti dalla posizione corrente, mentre -20 con OFFSET_END significa 20 byte prima della fine del file.

Per conoscere la posizione corrente del cursore senza modificarla si può richiamare Seek con un offset di zero rispetto alla posizione corrente.

Per muoversi alla fine del file si usa Seek con offset zero a partire dal termine. Per aggiungere informazioni alla fine di un file è possibile spostarsi alla fine del file con Seek e cominciare a scrivere da quel punto. Non si può spostare il puntatore oltre la fine del file.

SetComment

Scopo: aggiungere un commento.

Formato: *successo = SetComment(nome, commento)*
D0 D1 D2

Argomento: nome - puntatore alla stringa
commento - puntatore alla stringa

Risultato: successo - valore booleano

Descrizione: SetComment aggiunge un commento al file o alla directory "nome". Il commento è un puntatore a una stringa lunga fino a 80 caratteri e che termina con uno zero.

SetProtection

Scopo: impostare la protezione di un file o di una directory.

Formato: *successo = SetProtection(nome, maschera)*
D0 D1 D2

Argomento: nome - puntatore alla stringa
maschera - numero intero

Risultato: successo - valore booleano

Descrizione: SetProtection imposta gli attributi di protezione del file o della directory "nome". I quattro bit meno significativi della maschera hanno il seguente significato:

bit 3: se è a 1 non è permessa la lettura
bit 2: se è a 1 non è permessa la scrittura
bit 1: se è a 1 non è permessa l'esecuzione
bit 0: se è a 1 non è permessa la cancellazione

bit 4-31: riservati.

Nella versione attuale dell'AmigaDOS viene controllata solamente la protezione dalla cancellazione. Invece di riferirsi ai vari bit con il proprio numero d'ordine si possono usare le definizioni contenute in "include/libraries/dos.h".

UnLock

Scopo: rimuovere un lock associato a una directory o a un file.

Formato: *Unlock(lock)*
D1

Argomento: lock - puntatore a un lock

Descrizione: Unlock rimuove un lock ottenuto con una chiamata a Lock, DupLock oppure CreateDir.

WaitForChar

Scopo: indicare se un carattere giunge entro un certo limite di tempo.

Formato: *successo = WaitForChar(file, tempo)*
D0 D1 D2

Argomento: file - handle di un file
tempo - numero intero

Risultato: successo - valore booleano

Descrizione: se un carattere è disponibile per essere letto, entro un certo tempo indicato da "tempo", da un file associato all'handle "file", viene restituito -1 (VERO), altrimenti 0 (FALSO). Se un carattere è disponibile si può usare Read per leggerlo. WaitForChar, che non può essere usato in modo non interattivo, è valido solo se viene usato con canali di I/O che sono connessi a un terminale virtuale. "tempo" deve venir indicato in microsecondi.

Write

Scopo: scrivere byte di dati in un file.

Formato: *lunghezza_restituita = Write(file, buffer, lunghezza)*
D0 D1 D2 D3

Argomento: file - handle di un file
buffer - puntatore a un buffer
lunghezza - numero intero

Risultato: lunghezza_restituita - numero intero

Descrizione: si possono copiare dati usando una combinazione di Read e Write. Write scrive byte di dati nel file aperto associato all'handle "file"; "lunghezza" si riferisce alla quantità di dati da trasferire, mentre "buffer" è l'indirizzo di partenza del buffer.

Write restituisce un valore che indica il numero delle informazioni effettivamente scritte. Quando "lunghezza_restituita" è maggiore di zero indica il numero di caratteri

scritti, un valore di -1 indica un errore. Le routine che impiegano questa chiamata devono sempre accertarsi che non si presenti un errore, ad esempio quello che si verifica quando il disco è pieno.

Trattamento dei processi

CreateProc

Scopo: creare un nuovo processo.

Formato: *processo = CreatProc(nome, pri, segmento,*
D0 D1 D2 D3
dimensione_stack)
D4

Argomento: nome - puntatore alla stringa
pri - numero intero
segmento - puntatore a un segmento
dimensione_stack - numero intero

Risultato: processo - numero intero

Descrizione: CreateProc crea un nuovo processo dotato di nome "nome". CreateProc, in pratica, provvede ad allocare una struttura di controllo del processo in un'area della memoria disponibile, e la inizializza.

CreateProc riceve una lista di segmenti come argomento "segmento" (vedere anche LOADSEG e UNLOADSEG). Questa lista di segmenti rappresenta la sezione di codice che si intende eseguire come nuovo processo. CreateProc manda in esecuzione il codice presente nel primo segmento della lista, che dovrebbe contenere un'appropriata routine di inizializzazione oppure un salto a tale routine.

"dimensione_stack" rappresenta la dimensione in byte dello stack principale quando CreateProc attiva il processo. "pri" specifica la priorità richiesta per il nuovo processo. Il valore restituito è l'identificatore di processo del nuovo task, oppure zero se la routine è fallita.

L'argomento "nome" indica il nome del processo.

DateStamp

Scopo: ottenere la data e l'ora nel formato interno.

Formato: *DateStamp(v)*
D1

Argomento: v - puntatore al vettore

Descrizione: DateStamp accede a un vettore di tre long word che rappresenta la data attuale. Il primo elemento del vettore è il numero di giorni. Il secondo elemento è il numero di minuti passati dall'inizio del giorno, mentre il terzo è il numero di **tick** dall'inizio dell'attuale minuto. Un tick si verifica 50 volte al secondo. DateStamp assicura che i giorni e i minuti siano esatti. Tutti e tre gli elementi sono a zero se la data non è stata impostata. Attualmente DateStamp restituisce solamente multipli pari di 50 tick. Quindi l'ora che si ottiene è sempre un numero intero di secondi.

Delay

Scopo: sospendere un processo per un determinato periodo.

Formato: *Delay(tempo)*
D1

Argomento: tempo - numero intero

Descrizione: la funzione Delay richiede l'argomento "tempo" che indica per quanti tick (50 al secondo) il processo deve rimanere sospeso.

DeviceProc

Scopo: restituire l'identificatore del processo che gestisce l'I/O.

Formato: *processo = DeviceProc(nome)*
D0 D1

Argomento: nome - puntatore alla stringa

Risultato: processo - identificatore di processo

Descrizione: DeviceProc restituisce l'identificatore del processo che gestisce il device associato al nome. Se DeviceProc non riesce a trovare un processo che gestisce l'I/O, restituisce zero come risultato. Se "nome" si riferisce a un file presente in un device configurato, IoErr restituisce un puntatore a un lock di una directory.

È possibile usare questa funzione per determinare l'identificatore del processo verso il quale il sistema manda i propri messaggi.

Exit

Scopo: terminare un programma.

Formato: *Exit(codice_ritorno)*
D1

Argomento: codice_ritorno - numero intero

Descrizione: Exit agisce in maniera differente a seconda che si stia eseguendo il programma in ambiente CLI o meno. Se si sta eseguendo, come fosse un comando sotto CLI, un programma che richiama Exit, a quel punto l'esecuzione termina, e il controllo ritorna al CLI. Exit interpreta l'argomento "codice_ritorno" come un codice che viene restituito dal programma.

Se il programma viene eseguito come un processo distinto, Exit cancella il processo e rilascia tutto lo spazio occupato dallo stack, dalla lista dei segmenti e dalla struttura dati del processo.

Caricamento del codice

Execute

Scopo: eseguire un comando CLI.

Formato: *successo = Execute(comando, input, output)*
D0 D1 D2 D3

Argomento: comando - puntatore alla stringa
input - handle di un file
output - handle di un file

Risultato: successo - valore booleano

Descrizione: questa funzione riceve una stringa (comando) che indica un comando CLI completo dei vari argomenti, e cerca di eseguirla. La stringa CLI può contenere qualunque input valido che può essere digitato direttamente nel CLI, inclusi i simboli di ridirezione dell'input e dell'output < e >.

L'handle del file di input è normalmente zero e in questo caso la funzione Execute esegue quanto è specificato nella stringa di comando per poi restituire il controllo. Se l'handle del file di input non è zero, il comando specificato nella stringa legge il proprio input dal file caratterizzato da tale handle, finché non viene raggiunta la fine del file.

Nella maggior parte dei casi l'handle del file di output deve essere indicato, ed è usato dai comandi CLI per dirigere il proprio flusso di dati in output, a meno che non venga usata la ridirezione. Se l'handle del file di output è lasciato a zero allora viene usata la finestra corrente, normalmente indicata con *. Occorre notare che i programmi eseguiti sotto Workbench non dispongono in genere di una finestra corrente.

La funzione Execute può essere usata per creare un nuovo CLI interattivo, proprio come quello originato dal comando NEWCLI. Per fare questo si deve richiamare la funzione Execute con una stringa di comando vuota e passando l'handle del file di input relativo a una nuova finestra. L'handle del file di output deve essere a zero. Il CLI leggerà i comandi dalla nuova finestra che userà poi anche per l'output. Questa nuova finestra CLI può essere chiusa solo facendo uso di un comando ENDCLI. Affinché questo comando funzioni deve essere presente nella libreria C: il programma RUN.

LoadSeg

Scopo: caricare in memoria un modulo caricabile.

Formato: *segmento = LoadSeg(nome)*
D0 D1

Argomento: nome - puntatore alla stringa

Risultato: segmento - puntatore a un segmento

Descrizione: il file "nome" è un modulo caricabile prodotto dal linker. LoadSeg riceve questo modulo e lo carica sparpagliando i diversi segmenti di codice nella memoria e collegandoli insieme attraverso le loro prime word. Riconosce uno zero come indicatore della fine della catena.

Se si verifica un errore, LoadSeg rimuove tutti i blocchi caricati e restituisce un risultato pari a zero.

Se tutto procede a dovere, cioè se LoadSeg ha caricato tutti i moduli correttamente, viene restituito un puntatore all'inizio della lista dei blocchi. Una volta che si è terminato di usare il codice caricato, lo si deve rimuovere facendo uso di UnLoadSeg (per l'uso del codice caricato si veda CreateProc).

UnLoadSeg

Scopo: rimuovere i segmenti caricati in precedenza con LoadSeg.

Formato: *UnLoadSeg(segmento)*
D1

Argomento: segmento - puntatore a un segmento

Decrizione: UnLoadSeg rimuove gli identificatori di segmento che sono stati restituiti da LoadSeg. "segmento" può essere zero.

2.3 Tavola di consultazione rapida

Trattamento dei file

Close	per chiudere un file di input o output.
CreateDir	per creare una nuova directory.
CurrentDir	per rendere una directory associata a un lock la nuova directory corrente.
DeleteFile	per cancellare un file o una directory.
DupLock	per duplicare un lock.

Examine	per esaminare una directory o un file associato a un lock.
ExNext	per esaminare la successiva voce di una directory.
Info	per ottenere informazioni riguardanti un disco.
Input	per identificare l'handle del file di input iniziale.
IoErr	per ottenere dal sistema informazioni aggiuntive.
IsInteractive	per scoprire se un file è connesso o meno a un terminale virtuale.
Lock	per ottenere un lock di un file o di una directory.
Open	per aprire un file di input o di output.
Output	per identificare l'handle del file di output iniziale.
ParentDir	per ottenere il genitore di una directory o di un file.
Read	per leggere byte di dati da un file.
Rename	per cambiare il nome di una directory o di un file.
Seek	per muoversi a una posizione logica in un file.
SetComment	per aggiungere un commento.
SetProtection	per impostare la protezione di un file o di una directory.
UnLock	per rilasciare il lock di un file o di una directory.
WaitForChar	per indicare se un carattere giunge o meno entro un certo limite di tempo.
Write	per scrivere byte di dati in un file.

Trattamento dei processi

CreateProc	per creare un nuovo processo.
DateStamp	per ottenere la data e l'ora nel formato interno.
Delay	per sospendere un processo per un determinato periodo.
DeviceProc	per ottenere l'identificatore di un processo che gestisce l'I/O.
Exit	per terminare un programma.

Caricamento del codice

Execute	per eseguire un comando CLI.
LoadSeg	per caricare in memoria un modulo caricabile.
UnLoadSeg	per rimuovere un segmento precedentemente caricato con LoadSeg.

Capitolo 3

Il Macro Assembler

Questo capitolo descrive il Macro Assembler dell'AmigaDOS e fornisce una breve introduzione al microprocessore 68000. Si presuppone la conoscenza, da parte del lettore, delle nozioni di base del linguaggio Assembly.

3.1 Introduzione al microprocessore 68000
3.2 Richiamo dell'assemblatore
3.3 Codifica del programma
3.3.1 Commenti
3.3.2 Istruzioni eseguibili
3.3.2.1 Campo dell'etichetta
3.3.2.2 Etichette locali
3.3.2.3 Campo del codice dell'istruzione
3.3.2.4 Campo dell'operando
3.3.2.5 Campo del commento
3.4 Espressioni
3.4.1 Operatori
3.4.2 Gli operandi relativi agli operatori
3.4.3 Simboli
3.4.4 Numeri
3.5 Modi di indirizzamento
3.6 Variazioni sui tipi di istruzione
3.7 Direttive

3.1 Introduzione al microprocessore 68000

Questa sezione fornisce una breve introduzione al microprocessore 68000, e dovrebbe esservi d'aiuto per comprendere i concetti che saranno introdotti nel corso del capitolo.

La memoria a disposizione del 68000 è ripartita fra:

- i registri interni (contenuti nel chip)
- la memoria principale esterna.

Ci sono 17 registri, ma solo 16 sono sempre disponibili. Otto di questi, da D0 a D7, sono i **registri dei dati**, mentre gli altri otto, da A0 ad A7, sono i **registri di indirizzo**. Ogni registro è composto da 32 bit. In molti casi è possibile usare qualunque tipo di registro, ma altre volte bisogna usarne uno specifico. Per esempio, è possibile usare un qualunque registro per operazioni con **word** (16 bit) e **long word** (32 bit), oppure per l'indirizzamento indicizzato della memoria; per operazioni con byte (8 bit) si possono usare anche i registri dei dati, ma, per quanto riguarda l'indirizzamento della memoria, possono essere impiegati solo i registri di indirizzo come lo stack pointer o i registri base. Il registro A7 è lo stack pointer; di fatto è composto da due registri indipendenti: lo stack pointer di sistema, disponibile in modo supervisor, e lo stack pointer dell'utente, disponibile in modo user.

La memoria principale è costituita da un certo numero di byte. Ogni byte possiede un numero di identificazione chiamato **indirizzo**. La memoria di solito (ma non sempre) è organizzata in modo che i propri byte abbiano indirizzi 0, 1, 2, ..., N-2, N-1 dove N è il numero di byte totali della memoria. La quantità di memoria alla quale è possibile accedere direttamente è molto grande: fino a 16 milioni di byte. Il 68000 può eseguire operazioni con byte, word o long word. Una word corrisponde a due byte consecutivi, dei quali il primo possiede un indirizzo pari, mentre una long word è composta da quattro byte consecutivi che iniziano anch'essi a un indirizzo pari. L'indirizzo di una long word è l'indirizzo del primo byte che la compone.

Così come contiene i dati che vengono manipolati dal computer, la memoria contiene anche le **istruzioni** che comunicano al computer le operazioni che deve svolgere. Ogni istruzione occupa da una a cinque word, essendo costituita da una **word di operazione** e da zero a quattro word di operandi. La word di operazione indica quale azione deve essere intrapresa (implicitamente indica anche quante word ci sono nell'intera istruzione). Le **word di operandi** indicano dove si trovano i dati da trattare (quale registro o quale cella di memoria) e dove deve essere trasferito il risultato.

Il computer esegue di solito le istruzioni una alla volta, nello stesso ordine che occupano in memoria, così come si seguono le istruzioni di una ricetta o si suonano le note di un brano musicale, leggendole sul pentagramma. C'è un registro speciale chiamato **contatore di programma**, o più brevemente PC (dall'inglese Program Counter), che contiene l'indirizzo della successiva istruzione da eseguire. Alcune istruzioni, chiamate **salti** (jump) e **diramazioni** (branch), alterano il normale ordine di percorrenza e obbligano il processore a eseguire per prima l'istruzione memorizzata a un indirizzo specifico. Questo permette al computer di eseguire un'operazione ripetitiva-

mente, oppure di intraprendere compiti differenti a seconda del valore di certi dati.

Per conoscere lo stato del computer ci si può servire di un altro registro speciale chiamato **registro di stato** (anche SR, dall'inglese Status Register).

3.2 Richiamo dell'assemblatore

Il template del comando ASSEM è

"PROG=FROM/A,-O/K,-V/K,-L/K,-H/K,-C/K,-I/K"

In alternativa, il formato della linea di comando può essere descritto come

```
assem <file_sorgente>  [-o <file_oggetto>]
                       [-l <file_list>]
                       [-v <file_verifica>]
                       [-h <file_header>]
                       [-c <opzioni>]
                       [-i <dir_include>]
```

L'assemblatore non produce un file oggetto o un file list a meno che non venga richiesto esplicitamente.

Mentre è in esecuzione, l'assemblatore genera messaggi diagnostici (errori, avvertimenti e statistiche dell'assemblaggio) e li manda allo schermo se non è stato indicato un file di verifica.

Per forzare l'inclusione di un file in testa al file sorgente durante l'assemblaggio, si usa -h <filename> nella linea di comando. In questo modo si causa lo stesso effetto che si otterrebbe scrivendo

```
INCLUDE ''<filename>''
```

nella linea 1 del file sorgente.

Per costituire la lista di directory nelle quali l'assemblatore deve cercare i file da INCLUDEre si deve usare la keyword -i. Dopo -i si possono indicare tutte le directory che sono richieste avendo cura di separarle con una virgola (,), un simbolo di addizione (+) o uno spazio. Se si fa uso di uno spazio si deve includere l'intera lista di directory tra virgolette ("). Gli utenti dello Unix, devono comunque sostituire le virgolette con una barra inversa (\").

L'ordine della lista determina l'ordine nel quale le directory vengono scandite alla ricerca dei file INCLUDE. L'assemblatore effettua la sua ricerca iniziando dalla directory corrente. Quindi qualunque file da includere in un programma può trovarsi nella directory corrente oppure in una delle directory

presenti nella lista associata a -i. Per esempio, se il programma "marco" INCLUDE, senza contare i file presenti nella directory corrente, un file della directory "intrnl/incl", un altro appartenente a "include/asm" e uno di "extrnl/incl", è possibile fornire la lista di directory per -i in tre modi diversi:

```
assem marco -i intrnl/incl,include/asm,extrnl/incl
assem marco -i intrnl/incl+include/asm+extrnl/incl
assem marco -i ''intrnl/incl include/asm extrnl/incl''
```

oppure, usando lo spazio come separatore su un Sun in ambiente Unix

```
assem marco -i \''intrnl/incl include/asm extrnl/incl\''
```

La keyword -c permette di comunicare all'assemblatore alcune opzioni. Ogni opzione consiste di un singolo carattere (sia maiuscolo che minuscolo), immediatamente seguito eventualmente da un numero. Le opzioni valide sono le seguenti:

S produce una lista dei simboli come parte del file oggetto.
D evita che le etichette locali (local label) vengano comprese nella lista dei simboli (per i programmatori in C, qualunque etichetta che inizi con un punto è considerata un'etichetta locale).
C ignora la distinzione tra maiuscole e minuscole nelle etichette.
X produce una tavola di cross reference alla fine del file list.

Esempi

```
assem marco.asm -o marco.o
```

assembla il file "marco.asm" e produce il modulo oggetto, mettendolo nel file "marco.o".

```
assem marco.asm -o marco.o -l marco.lst
```

assembla il file "marco.asm" producendo il file oggetto "marco.o" e il file list "marco.lst".

3.3 Codifica del programma

Perché un programma risulti accettabile da parte dell'assemblatore deve avere la forma di una serie di linee di input che includono, secondo l'esigenza

specifica, le seguenti voci:

- Commenti o linee vuote
- Istruzioni eseguibili
- Direttive per l'assembler

3.3.1 Commenti

Per introdurre un commento in un programma si possono seguire tre diversi procedimenti:

1. Inserire un punto e virgola (;) in qualunque punto della linea e farlo seguire dal testo del commento. Per esempio

```
CMPA.L  A1,A2 ; i puntatori sono uguali?
```

2. Inserire un asterisco (*) nella prima colonna e farlo seguire dal testo del commento. Per esempio

```
* Tutta questa linea e' un commento
```

3. Far seguire una qualunque istruzione completa, o una direttiva, da almeno uno spazio e dal testo del commento. Per esempio

```
MOVEQ  #10,D0 pone un valore iniziale in D0
```

Inoltre tutte le linee vuote vengono trattate dall'assemblatore come linee di commenti.

3.3.2 Istruzioni eseguibili

Il formato generale di una istruzione sorgente è:

[<etichetta>] <codice_istruzione>
[<operando>[,<operando>]...][<commento>]

Per separare ogni campo dal successivo si deve usare un carattere di separazione, come uno spazio o un TAB. È possibile usare anche più di uno spazio per separare i diversi campi.

3.3.2.1 Campo dell'etichetta
Un'etichetta è un simbolo dell'utente, o meglio un nome definito dal

programmatore. Il campo che la contiene può
a) iniziare nella prima colonna ed essere separato dal campo successivo da almeno uno spazio
b) iniziare in qualunque colonna ed essere seguito immediatamente dai due punti (:).

Se è presente un'etichetta, questa dev'essere il primo elemento non vuoto della linea. L'assemblatore assegna all'etichetta il valore del contatore di programma, cioè l'indirizzo di memoria del primo byte dell'istruzione o dei dati ai quali si riferisce. Le etichette sono associabili a tutte le istruzioni e ad alcune direttive; inoltre possono trovarsi da sole su una linea. Per avere un elenco di tutti i casi specifici, riferirsi alle spiegazioni delle singole direttive nella sezione 3.7.
Nota: *non è possibile definire un' etichetta più volte e inoltre, i nomi delle etichette non possono essere nomi di istruzioni, di macro, di direttive o di registri.*

3.3.2.2 Etichette locali
Le etichette locali costituiscono un'estensione alle specifiche della Motorola. Sono caratterizzate dal formato nnn$ e sono valide soltanto nell'intervallo tra due etichette valide qualsiasi. Ad esempio, in questo segmento di codice:

Etichette	**Codici istruzioni**	**Operandi**
ABC:	MOVE.L	D6,D0
1$:	MOVE.B	(A0)+,(A1)+
	DBRA	D0,1$
	MOVEQ	#20,D0
XYZ:	TRAP	#4

l'etichetta 1$ è visibile solo dalla linea che segue quella contrassegnata da ABC fino a quella che ha come etichetta XYZ. In questo caso è possibile usare di nuovo l'etichetta 1$, per uno scopo diverso in qualunque altra parte del programma.

3.3.2.3 Campo del codice dell'istruzione
Il campo del codice dell'istruzione è situato dopo il campo dell'etichetta, dal quale deve essere separato con almeno uno spazio. Le voci inseribili in questo campo possono essere di tre tipi.

1. Il codice di una istruzione del MC68000, come viene definita nel *MC68000 User Manual.*
2. Una direttiva dell'assemblatore.
3. Una chiamata a una macro.

Per usare istruzioni o direttive che possono operare con dati di dimensioni diverse, si deve usare il campo opzionale che indica il formato. Questo deve essere separato dal codice dell'istruzione con un punto (.). I possibili indicatori sono:

B - dato composto da un Byte (8 bit)
W - dato composto da una Word (16 bit)
L - dato composto da una Long Word (32 bit)
oppure indicatore di una diramazione lunga (Long Branch)
S - indicatore di una diramazione corta (Short Branch)

Gli indicatori di formato devono essere coerenti con le istruzioni o con le direttive che vengono usate.

3.3.2.4 Campo dell'operando

Il campo dell'operando, se è presente, contiene uno o più operandi dell'istruzione o della direttiva, dalla quale deve essere separato con almeno uno spazio. Se sono presenti due o più operandi nel campo a loro riservato, devono essere separati per mezzo di una virgola (,). Il campo dell'operando termina con uno spazio o con un carattere che indica la fine della linea (corrisponde al carattere generato dalla pressione del tasto RETURN); quindi per separare gli operandi tra di loro non possono essere usati spazi.

3.3.2.5 Campo del commento

Qualunque cosa si trovi dopo lo spazio posto al termine del campo dell'operando viene ignorata. Quindi l'assemblatore tratta qualunque carattere inserito dopo lo spazio come un commento.

3.4 Espressioni

Un'espressione è una combinazione di simboli, costanti, operatori algebrici e parentesi che viene usata per indicare un valore finale nel campo dell'operando di un'istruzione o di una direttiva. Si possono includere nelle espressioni anche numeri relativi, ma possono essere usati solo con un insieme ristretto di operatori.

3.4.1 Operatori

Gli operatori disponibili sono elencati di seguito in ordine di precedenza.

1. Simbolo negativo, NOT logico (- e ~)
2. Shift sinistro, shift destro (<< e >>)

3. AND logico, OR logico (& e !)
4. Moltiplicazione, divisione (* e /)
5. Addizione, Sottrazione (+ e -)

Per alterare l'ordine di precedenza algebrico tra gli operatori, si devono racchiudere le espressioni tra parentesi. Quando gli operatori sono dotati di uguale precedenza, l'assemblatore esegue i calcoli da sinistra verso destra. Occorre notare che non si possono utilizzare spazi all'interno di un'espressione, visto che gli spazi vengono interpretati come separatori tra un campo e il successivo.

3.4.2 Gli operandi relativi agli operatori

Nella tavola seguente, che illustra le possibili combinazioni operando/operatore, e la relativa espressione risultante, "A" rappresenta un simbolo assoluto, "R" un numero relativo, mentre "x" indica un errore. Il segno meno e gli operatori logici sono validi solamente con operandi assoluti.

Operatori	Operandi			
	A op A	R op R	A op R	R op A
+	A	x	R	R
—	A	A	x	R
*	A	x	x	x
/	A	x	x	x
&	A	x	x	x
!	A	x	x	x
>>	A	x	x	x
<<	A	x	x	x

Tavola 3-A: gli operandi relativi agli operatori

3.4.3 Simboli

Un simbolo è una stringa formata da un massimo di 30 caratteri. Il primo

carattere di un simbolo dev'essere uno dei seguenti:

- Un carattere alfabetico maiuscolo o minuscolo (da "a" a "z" e da "A" a "Z").
- Un carattere di sottolineatura (_).
- Un punto (.).

I rimanenti caratteri della stringa possono appartenere alle precedenti categorie oppure essere numerici (da 0 a 9). In tutti i simboli, i caratteri minuscoli (a-z) *non* sono trattati come sinonimi dei caratteri maiuscoli equivalenti (a meno che non si usi l'opzione C dell'assemblatore). Quindi "giorgio" è diverso sia da "GIORGIO" che da "GIOrgio". Comunque l'assemblatore riconosce i codici delle istruzioni, le direttive e i nomi dei registri sia in minuscolo che in maiuscolo. Anche un'etichetta che venga posta uguale a un nome di registro per mezzo di EQUR, viene riconosciuta in qualunque forma sia scritta. I simboli possono essere lunghi fino a 30 caratteri, tutti significativi. L'assemblatore tronca i simboli più lunghi di 30 caratteri e genera un messaggio per informare l'utente della modifica effettuata. I nomi delle istruzioni, i nomi delle direttive, i nomi dei registri e i simboli speciali CCR, SR, SP e USP non possono essere usati come simboli dell'utente. Un simbolo può essere di tre tipi diversi:

Assoluto
a) Il simbolo viene impostato (direttiva SET) o posto uguale (direttiva EQU) a un valore assoluto.

Relativo
a) Il simbolo viene impostato (direttiva SET) o posto uguale (direttiva EQU) a un valore relativo.
b) Il simbolo è usato come un'etichetta.

Registro
a) Il simbolo viene impostato come un nome di registro usando la direttiva EQUR (è una estensione delle specifiche Motorola).

Il simbolo speciale "*" possiede lo stesso valore e lo stesso tipo del contatore di programma corrente, cioè l'indirizzo dell'istruzione o della direttiva che l'assemblatore sta elaborando.

3.4.4 Numeri

Un numero può essere usato come termine di un'espressione o come singolo valore. I numeri hanno SEMPRE un valore assoluto e possono

apparire in uno dei seguenti formati:

Decimale
(una stringa di cifre da 0 a 9)
Esempio: 1234

Esadecimale
($ seguito da una stringa di cifre esadecimali)
Esempio: $89AB

Ottale
(@ seguito da una stringa di cifre da 0 a 7)
Esempio: @743

Binario
(% seguito da una stringa di cifre da 0 a 1)
Esempio: %10110111

Literal ASCII
(fino a quattro caratteri ASCII racchiusi tra apostrofi)
Esempi: 'ABCD' '*'

Le stringhe composte da meno di quattro caratteri sono giustificate a destra, e vengono completate con caratteri nulli.
Per ottenere il carattere apostrofo in una stringa occorre usare due apostrofi consecutivi. Un esempio è:

```
'It''s'
```

3.5 Modi di indirizzamento

Il modo di indirizzamento utilizzato definisce il tipo di operando di un'istruzione o di una direttiva; una dettagliata trattazione di questo argomento può essere trovata in qualunque buon manuale sul 68000. Gli indirizzi si riferiscono a byte singoli, mentre le istruzioni, i riferimenti a word e long word accedono a più di un byte e l'indirizzo che li identifica deve essere allineato a una word.
Nella Tavola 3-B "Dn" rappresenta uno dei registri di dati (D0-D7), "An" uno dei registri di indirizzo (A0-A7, SP e PC), "a" rappresenta un'espressione assoluta, "r" un'espressione relativa, mentre "Xn" rappresenta sia An sia Dn, seguito dagli eventuali indicatori di formato ".W" e ".L". La sintassi per

ognuno dei modi è descritta nella tavola.

Modo d'indirizzamento	**Descrizione ed esempi**	
Dn	Diretto, registro dati	
	Esempio:	MOVE D0,D1
An	Diretto, registro indirizzo	
	Esempio:	MOVEA A0,A1
(An)	Indiretto tramite registro indirizzo	
	Esempio:	MOVE D0,(A1)
(An)+	Indiretto tramite registro indirizzo con post-incremento	
	Esempio:	MOVE (A7)+,D0
-(An)	Indiretto tramite registro indirizzo con pre-decremento	
	Esempio:	MOVE D0,-(A7)
a(An)	Indiretto tramite registro indirizzo con scostamento	
	Esempio:	MOVE 20(A0),D1
a(An,Xn)	Indiretto tramite registro indirizzo con indice	
	Esempi:	MOVE 0(A0,D0),D1
		MOVE 12(A1,A0.L),D2
		MOVE 120(A0,D6.W),D4
a	Assoluto corto (16 bit)	
	Esempio:	MOVE $1000,D0
a	Assoluto lungo (32 bit)	
	Esempio:	MOVE $10000,D0
r	Relativo al contatore di programma (PC) con scostamento	
	Esempio:	MOVE ABC,D0 (ABC è relativo)
r(Xn)	Relativo al contatore di programma (PC) con indice	
	Esempio:	MOVE ABC(D0.L),D1 (ABC è relativo)
#a	Dato immediato	
	Esempio:	MOVE #1234,D0
USP CCR SR	Modi di indirizzamento speciali	

	Esempi:	MOVE A0,USP
		MOVE D0,CCR
		MOVE D1,SR

Tavola 3-B: modi di indirizzamento e registri del macro assembler

3.6 Variazioni sui tipi di istruzione

Alcune istruzioni, per esempio ADD e CMP, dispongono di una **variazione di indirizzo** (che si riferisce ai registri di indirizzo come destinazioni), di una forma **immediata** e di una **veloce** (quando un dato immediato che appare come operando rimane entro un ristretto intervallo di valori) e infine di una **variazione di memoria** (entrambi gli operandi devono essere un indirizzo con post-incremento). Queste variazioni vengono indicate con delle lettere: "A" per **variazione d'indirizzo**, "I" per **immediata**, "Q" per **veloce**, e "M" per **variazione di memoria**.

Per forzare l'impiego di una particolare variazione si deve aggiungere all'istruzione mnemonica la lettera che la contraddistingue. L'assemblatore usa in questo caso la forma dell'istruzione specificata se esiste, altrimenti genera un messaggio di errore.

Se invece non si indica alcuna variazione particolare, l'assemblatore converte automaticamente le istruzioni nelle forme "I", "A" o "M" secondo la necessità. Comunque non opera nessuna conversione nella forma "Q". Per esempio l'assemblatore converte

```
ADD.L A2,A1
```

in

```
ADDA.L A2,A1
```

3.7 Direttive

Tutte le direttive assembler sono istruzioni dirette all'assemblatore, e non vengono trasformate in codice oggetto. In questa sezione si trova una lista di tutte le direttive (Tavola 3-C) organizzata secondo il tipo di funzione; al termine sono riportate le singole descrizioni di ogni direttiva.

Occorre notare che l'assemblatore permette l'uso di etichette in associazione a direttive solo nei casi indicati. Per esempio EQU può essere individuata da un'etichetta. Per RORG l'etichetta è opzionale, mentre non può essere usata con LLEN o TTL.

La tavola seguente illustra le direttive secondo la loro funzione.

Tavola 3-C: direttive

Controllo dell'assemblaggio

Direttiva	**Descrizione**
SECTION	Definisce una sezione di programma
RORG	Definisce l'origine rilocabile
OFFSET	Definisce gli offset
END	Termina il programma

Definizione di simboli

Direttiva	**Descrizione**
EQU	Assegna un valore in modo permanente
EQUR	Assegna un simbolo al nome di un registro
REG	Definisce una lista di registri
SET	Assegna un valore in modo temporaneo

Definizione di dati

Direttiva	**Descrizione**
DC	Definisce una o più costanti
DCB	Definisce blocchi di costanti
DS	Riserva memoria

Controllo del list

Direttiva	**Descrizione**
PAGE	Avanza alla pagina successiva
LIST	Abilita la generazione del list
NOLIST (NOL)	Disabilita la generazione del list
SPC n	Genera n linee vuote
NOPAGE	Disabilita la suddivisione in pagine
LLEN n	Imposta la lunghezza della linea ($60 \leq n \leq 132$)
PLEN n	Imposta la lunghezza della pagina ($24 \leq n \leq 100$)
TTL	Definisce il titolo del programma (massimo 40 caratteri)
NOOBJ	Disabilita la generazione del codice oggetto
FAIL	Genera un errore per il programmatore
FORMAT	Nessuna azione
NOFORMAT	Nessuna azione

Assemblaggio condizionato

Direttiva	**Descrizione**
CNOP	Genera NOP condizionati per l'allineamento
IFEQ	Assembla se l'espressione è uguale a 0
IFNE	Assembla se l'espressione è diversa da 0
IFGT	Assembla se l'espressione è maggiore di 0
IFGE	Assembla se l'espressione è maggiore o uguale a 0
IFLT	Assembla se l'espressione è minore di 0
IFLE	Assembla se l'espressione è minore o uguale a 0
IFC	Assembla se le stringhe sono identiche

Direttiva	**Descrizione**
IFNC	Assembla se le stringhe sono diverse
IFD	Assembla se il simbolo è definito
IFND	Assembla se il simbolo non è definito
ENDC	Termine dell'assemblaggio condizionato

Direttive macro

Direttiva	**Descrizione**
MACRO	Inizia una definizione di macro
NARG	Simbolo speciale
ENDM	Termina una definizione di macro
MEXIT	Esce dalla espansione di macro

Simboli esterni

Direttiva	**Descrizione**
XDEF	Definisce un'etichetta impiegabile esternamente
XREF	Definisce un nome esterno

Direttive generali

Direttiva	**Descrizione**
INCLUDE	Inserisce un file nel sorgente
MASK2	Nessuna azione
IDNT	Assegna un nome all'unità di programma

Direttive per il controllo dell'assemblaggio

SECTION Definisce una sezione di programma

Formato: [<etichetta>] SECTION <nome>[,<tipo>]

Descrizione: questa direttiva comunica all'assemblatore di riposizionare il contatore all'ultima locazione della sezione nominata, oppure a zero se è usata per la prima volta.

<nome>	è una stringa di caratteri eventualmente racchiusa tra virgolette.
<tipo>	se è presente deve essere una delle seguenti keyword:
CODE	indica che la sezione contiene codice rilocabile. Viene usato come default.
DATA	indica che la sezione contiene solamente dati inizializzati.
BSS	indica che la sezione contiene dati non inizializzati.

L'assemblatore può mantenere un massimo di 255 sezioni. Inizialmente l'assemblatore comincia con una sezione CODE priva di nome. L'assemblatore assegna al simbolo opzionale <etichetta> il valore del contatore di programma dopo che è stata eseguita la direttiva SECTION. In aggiunta, quando una sezione non è dotata di nome, le viene assegnata la keyword CODE.

RORG Definisce l'origine rilocabile
Formato: [<etichetta>] RORG <espass>
Descrizione: la direttiva RORG modifica il contatore di programma in modo che risulti essere spostato di <espass> byte rispetto all'inizio della sezione rilocabile attuale. L'assemblatore predispone locazioni di memoria rilocabili per le istruzioni seguenti, iniziando con il valore assegnato al contatore di programma. Per effettuare gli indirizzamenti nelle sezioni rilocabili occorre servirsi del modo d'indirizzamento "relativo al contatore di programma con scostamento". L'assegnazione del valore all'etichetta è identico a SECTION.

OFFSET Definisce gli offset
Formato: OFFSET <espass>
Descrizione: per definire una tavola di offset, che inizi all'indirizzo <espass>, attraverso le direttive DS, occorre fare ricorso alla direttiva OFFSET. I simboli definiti in una tavola di OFFSET sono mantenuti internamente, ma non possono apparire né direttive, né istruzioni che producano codice. Per porre termine a una sezione OFFSET si usano le direttive RORG, OFFSET, SECTION oppure END.

END Termina il programma
Formato: [<etichetta>] END
Descrizione: la direttiva END comunica all'assemblatore che il sorgente è finito; di conseguenza le successive istruzioni vengono ignorate. Quando l'assemblatore incontra la direttiva END durante la prima fase, dà subito inizio alla seconda fase. Se invece incontra la fine del file prima della direttiva END, genera un messaggio di avvertimento. Se l'etichetta è presente, le viene assegnato il valore del contatore di programma prima dell'esecuzione della direttiva END.

Direttive per la definizione dei simboli

EQU Assegna un valore al simbolo in modo permanente

Formato: <etichetta> EQU <espressione>

Descrizione: la direttiva EQU assegna il valore dell'espressione posta nel campo dell'operando al simbolo che si trova nel campo dell'etichetta. Il valore assegnato è permanente e di conseguenza il simbolo non può essere nuovamente definito nel resto del programma.

Nota: *non si devono inserire nell'espressione riferimenti a parti successive del file.*

EQUR Assegna un simbolo al nome di un registro

Formato: <etichetta> EQUR <registro>

Descrizione: questa direttiva permette di assegnare un simbolo dell'utente a uno dei registri del processore. Solo i registri dei dati e degli indirizzi sono validi; quindi i simboli speciali come SR, CCR e USP sono illegali in questo contesto. Il registro è permanente e di conseguenza l'etichetta non può essere definita nuovamente in nessun'altra parte del programma. <registro> non può riferirsi a una successiva direttiva EQUR nel file. L'assemblatore confronta le etichette definite in questa maniera senza distinguere tra maiuscolo e minuscolo.

REG Definisce una lista di registri

Formato: <etichetta> REG <lista_registri>

Descrizione: la direttiva REG assegna un'etichetta a una lista di registri, in modo che l'assemblatore la possa trasformare nel formato di mascheramento della lista di registri usato dall'istruzione MOVEM (questa istruzione del 68000 salva sullo stack una lista di registri). <lista_registri> è nella forma

R1 [-R2][/R3[-R4]]...

dove il trattino individua i due delimitatori di una possibile serie di registri, mentre la barra divide fra loro i gruppi di registri.

SET Assegna un valore al simbolo in modo temporaneo

Formato: <etichetta> SET <espressione>

Descrizione: la direttiva SET assegna il valore dell'espressione presente nel campo dell'operando al simbolo posto nel campo dell'etichetta. La direttiva SET è identica a EQU, eccezion fatta per l'assegnamento che è temporaneo. Si può sempre modificare SET in un'altra parte del programma.

Nota: *non si devono inserire riferimenti a parti successive del file all'interno dell'espressione, né a simboli definiti successivamente con SET.*

Direttive per la definizione di dati

DC Definisce una o più costanti

Formato: [<etichetta>] DC[.<quantità>] <lista>

Descrizione: la direttiva DC definisce in memoria valori costanti. Accetta un qualunque numero di operandi separati da virgole (,). I valori presenti nella lista devono poter essere contenuti nelle locazioni dei dati la cui dimensione è determinata dall'argomento <quantità> della direttiva. Se non viene fornita alcuna dimensione, DC assume che sia .W. Se la quantità scelta è .B può essere usato un altro tipo di dato: una stringa ASCII. Una stringa è una serie di caratteri ASCII lunga quanto si desidera, racchiusa tra virgolette. Così come per i literal ASCII, se si richiede la presenza di una virgoletta all'interno della stringa, se ne devono scrivere due. Se la quantità è .W o .L, l'assemblatore allinea i dati secondo i limiti delle word.

DCB Definisce blocchi di costanti

Formato: [<etichetta>]DCB[.<quantità>]
<esp_ass>,<espressione>

Descrizione: si usa la direttiva DCB per definire un certo numero, indicato da <esp_ass>, di byte, word o long word con il valore dell'espressione <espressione>. Scrivere DCB.<quantità> n,espressione è equivalente a ripetere n volte la direttiva DC.<quantità> espressione.

DS Riserva memoria

Formato: [<etichetta>] DS[.<quantità>] <esp_ass>

Descrizione: per riservare locazioni di memoria si usa la direttiva DS che, comunque, non opera alcuna inizializzazione. La quantità di spazio allocata dall'assemblatore dipende dalla dimensione del dato (che viene fornita con l'indicatore di formato della direttiva) e dal valore dell'espressione contenuta nel campo dell'operando. L'assemblatore interpreta questa espressione come il numero di dati di quella dimensione da allocare. Come con DC, se l'indicatore è .W o .L, DS allinea lo spazio rispetto a un limite di word. Così, DS.W 0 sortisce solo l'effetto di un allineamento a un limite di word. Se non viene specificato l'indicatore di formato, viene assunto per default .W. Riferirsi a CNOP per un modo più generale di trattare l'allineamento.

Direttive per il controllo del list

PAGE Avanza alla pagina successiva
Formato: PAGE
Descrizione: a meno che la suddivisione in pagine sia stata disattivata, PAGE avanza il list dell'assemblaggio all'inizio della pagina successiva. La direttiva PAGE non appare nel list di output.

LIST Abilita la generazione del list
Formato: LIST
Descrizione: la direttiva LIST comunica all'assemblatore di produrre un file di list dell'assemblaggio. La generazione di questo file prosegue finché non viene incontrata una direttiva END oppure NOLIST. Questa direttiva non appare nel list di output.

NOLIST Disabilita la generazione del list
Formato: NOLIST
NOL
Descrizione: la direttiva NOLIST o NOL ferma la produzione del file di list dell'assemblaggio. La generazione è inattiva finché non viene incontrata una direttiva END oppure una LIST. La direttiva NOLIST non appare nel list di output.

SPC Genera linee vuote
Formato: SPC <numero>
Descrizione: la direttiva SPC, che non appare nel list di output, genera nel file di list tante linee vuote quante sono quelle indicate dal campo dell'operando.

NOPAGE Disabilita la suddivisione in pagine
Formato: NOPAGE
Descrizione: la direttiva NOPAGE disabilita i salti di pagina (direttiva PAGE) e i titoli d'intestazione nel list dell'assemblaggio.

LLEN Imposta la lunghezza della linea
Formato: LLEN <numero>
Descrizione: la direttiva LLEN assegna come lunghezza delle linee del file di list dell'assemblaggio il valore indicato nel campo dell'operando. Il valore deve essere compreso tra 60 e 132 e può essere assegnato una sola volta nel corso del programma. La direttiva LLEN non appare nel list di output. Il default è 132 caratteri.

PLEN Imposta la lunghezza della pagina
Formato: PLEN <numero>
Descrizione: la direttiva PLEN assegna come lunghezza della pagina del file di list dell'assemblaggio, il valore indicato nel campo dell'operando. Il valore deve essere compreso tra 24 e 100 e può essere assegnato una sola volta nel corso del programma. Il default è 60 linee.

TTL Definisce il titolo del programma
Formato: TTL <stringa>
Descrizione: la direttiva TTL definisce come titolo del programma la stringa presente nel campo dell'operando. La stringa apparirà come intestazione di pagina nel list dell'assemblaggio. L'inizio della stringa si identifica con il primo carattere non nullo dopo TTL e termina con la fine della linea. Non può essere più lunga di 40 caratteri. La direttiva TTL non appare nel list di output.

NOOBJ Disabilita la generazione del codice oggetto
Formato: NOOBJ
Descrizione: la direttiva NOOBJ disabilita la generazione del codice oggetto al termine dell'assemblaggio, anche se facendo ricorso all'assemblatore si era specificato un file di output.

FAIL Genera un errore per il programmatore
Formato: FAIL
Descrizione: la direttiva FAIL comunica all'assemblatore di segnalare un errore per questa linea.

FORMAT Nessuna azione
Formato: FORMAT
Descrizione: l'assemblatore accetta questa direttiva, per mantenere la compatibilità con altri assemblatori, ma non esegue nessuna operazione.

NOFORMAT Nessuna azione
Formato: NOFORMAT
Descrizione: l'assemblatore accetta questa direttiva, per mantenere la compatibilità con altri assemblatori, ma non esegue nessuna operazione.

Direttive per l'assemblaggio condizionato

CNOP Genera NOP condizionati per l'allineamento

Formato: [<etichetta>] CNOP <numero>,<numero>

Descrizione: questa direttiva è un'estensione rispetto allo standard Motorola e permette di allineare in qualunque modo una sezione di codice. In particolare permette che qualunque struttura o punto di entrata di una funzione sia allineato in una long word.

La prima espressione rappresenta un offset, mentre la seconda indica l'allineamento richiesto per la base. Il codice viene allineato all'offset specificato partendo dall'allineamento più vicino. Quindi

```
CNOP 0,4
```

allinea il codice alla successiva long word, mentre

```
CNOP 2,4
```

allinea il codice alla word che si trova 2 byte oltre la prima long word.

IFEQ Assembla se espressione = 0
IFNE Assembla se espressione ≠ 0
IFGT Assembla se espressione > 0
IFGE Assembla se espressione ≥ 0
IFLT Assembla se espressione < 0
IFLE Assembla se espressione ≤ 0

Formato: IFxx <espressione>

Descrizione: si usano le direttive del tipo IFxx per abilitare o disabilitare l'assemblaggio a seconda del valore dell'espressione presente nel campo dell'operando. Se la condizione non è vera (per esempio, IFEQ 2+1), l'assemblaggio termina (cioè viene disabilitato). L'ordine di non assemblare rimane attivo finché l'assemblatore non raggiunge la direttiva ENDC corrispondente. Si possono nidificare tanti livelli di assemblaggio condizionato quanti ne occorrono, purché si termini ogni livello con un corrispondente ENDC.

IFC Assembla se le stringhe sono identiche
IFNC Assembla se le stringhe sono diverse

Formato: IFC <stringa>,<stringa>
IFNC <stringa>,<stringa>

Descrizione: le stringhe devono essere una serie di caratteri ASCII racchiusi tra apostrofi, per esempio 'abc' o '' (la stringa nulla). Se la condizione non è vera viene interrotto l'assemblaggio. Analogamente a quanto già visto, questa condizione rimane attiva finché non viene raggiunta la direttiva ENDC corrispondente.

IFD Assembla se il simbolo è definito
IFND Assembla se il simbolo non è definito
Formato: IFD <simbolo>
IFND <simbolo>
Descrizione: a seconda che il simbolo sia già stato definito o meno, l'assemblatore abilita o disabilita l'assemblaggio finché non incontra una ENDC corrispondente.

ENDC Termine dell'assemblaggio condizionato
Formato: ENDC
Descrizione: per terminare l'assemblaggio condizionato, attivato con una delle otto direttive IFxx, si usa la direttiva ENDC. ENDC è sempre riferita alla direttiva condizionale incontrata per ultima.

Direttive per le macro

MACRO Inizia una definizione di macro
Formato: <etichetta> MACRO
Descrizione: MACRO individua l'inizio di una definizione di macro, mentre ENDM ne individua il termine. Si deve indicare un'etichetta, che viene usata dall'assemblatore come nome della macro; le ricorrenze seguenti di questa etichetta come operando causano l'espansione del contenuto della macro e il suo inserimento all'interno del codice sorgente. Una macro può contenere qualunque codice d'istruzione, quasi tutte le direttive dell'assemblatore e ogni altra macro che sia già stata definita. Un simbolo d'addizione (+) nel file list indica qualunque codice generato da un'espansione di macro. Dopo il nome di una macro possono essere aggiunti una serie di argomenti separati da virgole. Se un argomento contiene uno spazio (una stringa, per esempio, può contenere uno spazio), l'intero argomento deve essere racchiuso tra il simbolo di minore (<) e di maggiore (>).

L'assemblatore memorizza e salva il codice sorgente che

riceve, dopo una direttiva MACRO e prima di una ENDM, assumendolo come macro istruzione. Il codice di una macro può contenere qualunque codice sorgente; inoltre il simbolo di barra inversa (\ ) possiede un significato speciale. La barra inversa seguita da un numero "n" indica che il valore dell'ennesimo argomento deve essere inserito nel codice in sua vece. Se l'ennesimo argomento non esiste non viene inserito nulla. La barra inversa seguita dal simbolo "@" indica all'assemblatore di generare il testo ".nnn", dove nnn è il numero di volte che la combinazione \@ è stata incontrata. Questa caratteristica viene normalmente usata per generare label uniche all'interno di una macro.

Non si possono nidificare le definizioni di macro, cioè non si può definire una macro all'interno di un'altra, ma si può richiamare una macro definita in precedenza. Comunque esiste un limite - attualmente è dieci - alla nidificazione delle chiamate di macro.

L'espansione di una macro termina quando l'assemblatore incontra la fine del testo memorizzato della macro, oppure quando incontra la direttiva MEXIT.

NARG Simbolo speciale
Formato: NARG
Descrizione: il simbolo NARG è un simbolo speciale riservato, al quale l'assemblatore assegna l'indice dell'ultimo parametro passato alla macro nella lista dei parametri. All'esterno di un'espansione di macro, NARG ha valore 0.

ENDM Termina una definizione di macro
Formato ENDM
Descrizione: la direttiva conclude una definizione di macro iniziata da una direttiva MACRO.

MEXIT Esce da un'espansione di macro
Formato: MEXIT
Descrizione: questa direttiva di solito si usa in unione alle direttive IFEQ e IFNE, per uscire dalla modalità di espansione delle macro, permettendo quindi l'espansione condizionata delle macro. Una volta che tale direttiva è stata eseguita, l'assemblatore interrompe l'espansione della macro come se non ci fosse altro testo memorizzato da includere.

Simboli esterni

XDEF Definisce un'etichetta interna come punto d'entrata esterno
Formato: XDEF <etichetta>[,<etichetta>...]
Descrizione: la direttiva XDEF è seguita da una o più etichette rilocabili. Ogni etichettta qui definita genera una definizione di simbolo esterno. Si possono operare riferimenti a questo simbolo da altri moduli, scritti anche in un linguaggio ad alto livello; il linker provvede poi a realizzare gli opportuni riferimenti. Se si usa questa direttiva oppure XREF, non si può eseguire direttamente il codice prodotto dall'assemblatore.

XREF Definisce un nome esterno al sorgente
Formato: XREF <etichetta>[,<etichetta>...]
Descrizione: la direttiva XREF è seguita da una o più etichette che non sono state definite in nessun'altra parte del programma. Gli usi successivi di una di queste etichette fanno generare all'assemblatore un riferimento esterno all'etichetta in questione. L'etichetta si usa come se ci si riferisse a un valore assoluto o rilocabile, a seconda che la corrispondente XDEF si riferisca a un simbolo assoluto o rilocabile.

Il linker ricava i valori corretti per l'uso accedendo a un altro modulo. Inoltre, genera qualunque informazione di rilocazione necessaria perché il codice risultante sia rilocabile.

I simboli esterni sono normalmente usati come segue. Per indicare una routine in un segmento di programma come definizione esterna, si pone un'etichetta all'inizio della routine e la si cita anche in una direttiva XDEF. Un altro programma può richiamare questa routine se dichiara l'etichetta tramite una direttiva XREF, "saltando" così alla routine.

Direttive generali

INCLUDE Inserisce un file nel sorgente
Formato: INCLUDE "<filename>"
Descrizione: la direttiva INCLUDE permette di inserire un file esterno nel programma sorgente. Il file che INCLUDE inserisce è definito dalla stringa posta nel campo dell'operando. Si possono nidificare le direttive INCLUDE fino a una profondità di tre, racchiudendo i filename tra virgolette come mostrato. INCLUDE risulta particolarmente utile quando un

insieme standard di definizioni di macro e di direttive EQU è richiesto in più programmi.

Si possono includere le definizioni in un singolo file per riferirsi a esse da altri programmi con un'appropriata direttiva INCLUDE. Conviene spesso porre le direttive NOLIST e LIST al principio e al termine dei file che si intendono inserire per mezzo di INCLUDE. L'AmigaDOS ricerca il file indicato prima nella directory corrente, poi in ogni directory presente nella lista fornita di seguito all'opzione -i.

MASK2 Nessuna azione

Formato: MASK2

Descrizione: l'assemblatore accetta la direttiva MASK2 ma non esegue nessuna operazione.

IDNT Assegna un nome all'unità di programma

Formato: IDNT <stringa>

Descrizione: un'unità di programma di una o più sezioni, deve avere un nome. Usando la direttiva IDNT si può definire il nome, consistente in una stringa eventualmente racchiusa tra virgolette. Se l'assemblatore non trova una direttiva IDNT, viene usato come nome dell'unità di programma una stringa nulla.

Capitolo 4

Il linker

Questo capitolo descrive il linker dell'AmigaDOS, che produce un singolo file binario caricabile da uno o più file di input ed è inoltre in grado di produrre programmi in overlay.

4.1 Introduzione
4.2 Uso del linker
4.2.1 Sintassi della linea di comando
4.2.2 File WITH
4.2.3 Errori e altre eccezioni
4.2.4 Output MAP e XREF
4.3 Gestione degli overlay
4.3.1 Direttiva OVERLAY
4.3.2 Riferimenti ai simboli
4.3.3 Precauzioni e avvertimenti
4.4 Codici e messaggi d'errore

4.1 Introduzione

ALINK produce un singolo file binario di output partendo da uno o più file di input. Questi file di input, conosciuti come **file oggetto**, possono contenere informazioni relative a simboli esterni. Per produrre i file oggetto si usa un assemblatore o un compilatore. Prima di produrre l'output, il **file caricabile**, il linker risolve tutti i riferimenti ai simboli.

Il linker può inoltre originare una mappa del link e una tavola di riferimento incrociato (cross reference) dei simboli.

Associato al linker c'è un **supervisore degli overlay** che può essere usato per sovrapporre in memoria i programmi di grandi dimensioni. Il linker produce i file caricabili adatti per operare in overlay. L'overlay è un sistema che consente a un programma principale di richiamare in memoria sotto-programmi secondo le necessità.

Il linker può essere usato in due modi diversi:

1. Per **linea di comando**. La maggior parte delle informazioni necessarie all'esecuzione del linker sono specificabili come parametri del comando.

2. Per **file di parametri**. Come alternativa, se un programma deve subire un link in modo ripetitivo, è possibile usare un file di parametri per specificare i dati necessari al linker.

Questi due metodi accettano tre tipi di file di input:

1. **Input binario primario**. Si riferisce a uno o più file oggetto che formano l'input binario iniziale del linker. Questi file sono sempre trasferiti nel file caricabile; l'input primario non può essere vuoto.
2. **File di overlay**. Se si deve operare in overlay, l'input primario forma la radice dell'albero delle sovrapposizioni e i file sovrapponibili fanno parte del resto della struttura.
3. **Librerie**. Si riferisce al codice specifico che il linker incorpora automaticamente. Le librerie possono essere residenti o associate. Una **libreria residente** è un file caricabile che può risiedere in memoria, oppure può venir caricato da una chiamata alla funzione dell'Exec "Open Library". Una **libreria associata** è un file oggettocontenuto in un file di archiviazione. Il linker carica il file soltanto se esiste un riferimento esterno alla libreria non ancora risolto.

L'opera del linker si svolge in due fasi.

1. Durante la prima fase il linker legge tutti i file primari, le librerie e gli overlay, registrando i segmenti di codice e le informazioni riguardanti i riferimenti esterni. Al termine il linker è in grado di generare, se ne è stata fatta richiesta, la mappa e la tavola di riferimento incrociato.
2. Se è stato indicato un file di output, il linker procede alla seconda fase scandendo nuovamente l'input. Per prima cosa copia i file primari di input nell'output risolvendo nel contempo i riferimenti ai simboli, poi aggiunge i necessari segmenti di codice delle librerie nello stesso modo. Si può quindi notare che i segmenti di codice delle librerie fanno parte della radice dell'albero delle sovrapposizioni. Infine il linker produce i dati per il supervisore degli overlay e trasferisce in output i file sovrapponibili.

Durante il primo passaggio, dopo che sono stati letti i file primari e quelli in overlay, il linker ispeziona la propria tavola dei simboli e, se trova qualche riferimento rimasto irrisolto, legge gli eventuali file indicati come librerie di input. In seguito il linker contrassegna qualunque segmento di codice contenente le definizioni esterne dei riferimenti irrisolti per includerlo in seguito nel file caricabile. Il linker inserisce solo i segmenti di codice delle librerie nei quali sono presenti dei riferimenti.

4.2 Uso del linker

Per usare il linker è necessario conoscere la sintassi del comando, il tipo di input e di output adottato e i possibili errori ai quali si può andare incontro. Questi argomenti vengono affrontati nei paragrafi successivi.

4.2.1 Sintassi della linea di comando

Il comando ALINK ha i seguenti parametri

ALINK [FROM|ROOT] files [TO file] [WITH file] [VER file] [LIBRARY|LIB files] [MAP file] [XREF file] [WIDTH n]

Il template delle keyword è

"FROM=ROOT,TO/K,WITH/K,VER/K,LIBRARY=LIB/K,MAP/K,XREF/K, WIDTH/K"

Nella sintassi del comando illustrata in precedenza, "file" indica il nome di un singolo file, "files" indica nessun nome o più nomi di file, separati da una virgola (,) o da simbolo d'addizione (+); "n" è un numero intero.
Di seguito sono riportati alcuni esempi di usi validi del comando ALINK:

```
ALINK a
ALINK ROOT a+b+c+d MAP file-map WIDTH 120
ALINK a,b,c TO output LIBRARY :flib/lib,obj/newlib
```

Il linker legge, nell'ordine indicato, le liste di file che gli vengono fornite. I parametri hanno i seguenti significati:

FROM: indica i file oggetto che si desidera appartengano all'input binario primario. Il linker copia sempre il contenuto di tali file nel file caricabile per formare una parte della radice degli overlay. Almeno un file deve essere indicato come input primario. ROOT è un sinonimo di FROM.

TO: specifica la destinazione per il file caricabile. Se questo parametro non esiste, il linker non effettua il secondo passo.

WITH: indica il file contenente i parametri del linker, per esempio una linea di comando normale. Di solito con questo parametro si usa un solo file, ma per completezza è possibile fornirne una lista.

Occorre notare che i parametri riportati nella linea di comando sostituiscono quelli eventualmente presenti nel file WITH. Nella sezione che segue è possibile trovare una completa descrizione della sintassi di questi file.

VER: indica la destinazione dei messaggi generati dal linker. Se non si indica VER, il linker manda i messaggi verso l'output standard (solitamente il terminale).

LIBRARY: specifica i file che si desidera vengano scanditi come librerie. Il linker include soltanto i segmenti di codice con riferimenti. LIB è un sinonimo di LIBRARY.

MAP: indica la destinazione della mappa di link.

XREF: indica la destinazione della tavola di riferimento incrociato.

WIDTH: dichiara la larghezza dell'output che il linker può usare quando produce la mappa di link e la tavola di riferimento incrociato. Per esempio, se si manda l'output a una stampante, è possibile che sia necessario ricorrere a questo parametro.

4.2.2 File WITH

I file WITH contengono i parametri per il linker e vengono usati per evitare continue digitazioni di complesse linee di comando durante l'attivazione di ALINK.

Un file WITH consiste in una serie di parametri, uno per linea, composti da una keyword seguita dai dati. Si possono terminare le linee con un punto e virgola (;). Il linker ignora quanto si trova oltre il punto e virgola, rendendo così disponibile uno spazio per inserire eventuali commenti. Il linker ignora anche le linee vuote.

Le keyword disponibili sono le seguenti:

```
FROM (o ROOT)     files
TO                file
LIBRARY           files
MAP               [file]
XREF              [file]
OVERLAY
specifica dell'albero
#
WIDTH             n
```

dove "file" è un singolo filename, "files" è uno o più filename, "[file]" è un filename opzionale e "n" è un intero. Si può usare un asterisco (*) per suddividere le linee troppo lunghe. Basta porne uno al termine di una linea perché quella successiva venga considerata come il proseguimento di quella precedente e non come una nuova linea. Se il filename posto dopo MAP o XREF viene omesso, l'output viene inviato verso il file VER (che per default è il terminale).

I parametri sulla linea di comando sostituiscono i corrispondenti parametri posti nel file WITH, così si possono operare piccole variazioni ai link standard combinando i parametri della linea di comando con i file WITH. Analogamente, se si indica un parametro più di una volta nei file WITH, il linker usa il primo che ha incontrato, anche se è nullo.

Di seguito sono riportati alcuni esempi di file WITH e delle corrispondenti chiamate ad ALINK.

```
ALINK WITH file-link
```

dove "file-link" contiene

```
FROM       obj/main,obj/s
TO         bin/test
LIBRARY    obj/lib
MAP
XREF       xo
```

è lo stesso che indicare

```
ALINK FROM obj/main,obj/s TO bin/test LIBRARY obj/lib XREF xo
```

Il comando

```
ALINK WITH lkin LIBRARY ''''
```

nel quale "lkin" contiene

```
FROM       bin/prog,bin/subs
LIBRARY    nag/fortlib
TO         linklib/prog
```

è uguale a

```
ALINK FROM bin/prog,bin/subs TO linklib.prog
```

Nota: *nell'esempio precedente il parametro nullo di LIBRARY della linea di comando sostituisce il valore "nag/fortlib" del file WITH e quindi il linker non accede ad alcuna libreria.*

4.2.3 Errori e altre eccezioni

Mentre il linker è in esecuzione si possono verificare vari errori. La maggior parte dei messaggi sono autoesplicativi e si riferiscono al fallimento nell'apertura dei file, a errori nei comandi o al formato dei file binari. Dopo un errore il linker termina lo svolgimento delle operazioni.

Ci sono alcuni messaggi che sono solo avvertimenti; il più importante di questi si riferisce ai simboli non definiti oppure definiti più volte. Il linker non termina le operazioni se genera un avvertimento.

Se alla fine del primo passo esistono ancora dei simboli non definiti, viene generato un avvertimento e viene mostrata una tavola contenente questi simboli. Durante il secondo passo i riferimenti ai simboli non definiti divengono riferimenti alla **locazione zero**.

Se il linker trova un simbolo definito più volte, durante il primo passo, genera un avvertimento e ignora l'ultima definizione. Se la seconda definizione si verifica in file di libreria non viene generato il messaggio, in modo che si possano sostituire le routine delle librerie senza produrre messaggi spuri. Se il linker trova riferimenti a simboli inconsistenti, viene generato un errore e di conseguenza termina la fase di link.

Dal momento che il linker usa soltanto la prima definizione di un simbolo, è importante capire in quale ordine i file vengono letti:

1. Input primario (FROM oppure ROOT).
2. File overlay.
3. File di librerie (LIBRARY).

All'interno di ogni gruppo, i file vengono letti nell'ordine indicato dalla lista fornita. Così le definizioni nell'input primario hanno la priorità rispetto a quelle dei file overlay e quelle nelle librerie dispongono della priorità più bassa.

4.2.4 Output MAP e XREF

La mappa di list che il linker genera dopo il primo passaggio, contiene una lista di tutti i segmenti di codice che sono stati passati in output nel file caricabile, nell'ordine in cui sono stati scritti.

Per ogni segmento di codice viene visualizzata un'intestazione che inizia con il nome del file (troncato a otto lettere), il numero di riferimento del segmento di codice, il tipo (cioè DATA, CODE, BBS o COMMON) e la dimensione. Se il segmento di codice è in un file overlay, il linker mostra anche il livello e l'ordinata di sovrapposizione.

Dopo l'intestazione il linker stampa ogni simbolo definito nel segmento di codice, insieme al proprio valore. I simboli vengono mostrati in ordine crescente rispetto ai loro valori, aggiungendo un asterisco ai valori assoluti.
Il valore di WIDTH determina il numero di simboli stampati su una linea: se è troppo piccolo, il linker mostra un solo simbolo per linea.
Anche l'output di riferimento incrociato mostra ogni segmento di codice, utilizzando la stessa intestazione usata dalla mappa di link.
L'intestazione è seguita da una lista dei simboli con i propri riferimenti, che consistono in una coppia di interi, uno indicante l'offset del riferimento e l'altro il numero del segmento di codice nel quale si è verificato. Il numero del segmento di codice si riferisce al numero fornito in ogni intestazione.

4.3 Gestione degli overlay

Il sistema automatico di sovrapposizione fornito dal linker e dal supervisore degli overlay permette una minore occupazione di memoria da parte dei programmi quando questi ultimi sono in esecuzione, senza nessuna alterazione della loro struttura.
Quando si usano gli overlay bisogna considerare il programma come una struttura ad **albero** avente come **radice** l'input binario primario insieme ai segmenti di librerie e ai blocchi COMMON. La radice risiede costantemente in memoria. I file overlay formano gli altri nodi dell'albero, in accordo con quanto specificato nella direttiva OVERLAY.
L'output del linker che viene prodotto con la sovrapposizione è, come al solito, un file binario singolo costituito da tutti i segmenti di codice, uniti alle informazioni che forniscono le locazioni di ogni nodo dell'albero all'interno del file. Quando il programma viene caricato, soltanto la radice viene effettivamente trasferita in memoria. Un supervisore degli overlay ha il compito di caricare e rimuovere automaticamente i vari segmenti. Il linker aggiunge questo supervisore, che risulta essere trasparente al programma in esecuzione, ogni volta che effettua un link che necessiti di file overlay.

4.3.1 Direttiva OVERLAY

Per indicare al linker la struttura ad albero di un programma si usa la direttiva OVERLAY, che può essere usata solo nei file WITH. Il linker, come per gli altri parametri, usa la prima direttiva OVERLAY che incontra.

Il formato della direttiva è

OVERLAY
Xfile

#

Nota: *la direttiva OVERLAY può occupare più di una linea. Il linker riconosce la fine della direttiva dalla presenza del simbolo di numero (#) oppure dalla fine del file.*

Ogni linea dopo OVERLAY indica un nodo dell'albero che consiste nel contatore "X" e in una lista di file.

Il livello di un nodo indica la sua "profondità" nell'albero, partendo da zero, che è il livello della radice. Il contatore "X" fornito nella direttiva è composto da zero o più asterischi, mentre il livello di sovrapposizione del nodo è dato da X+1.

Così come il livello, ogni nodo diverso dalla radice possiede un valore di **ordinata**. Si riferisce all'ordine nel quale il linker dovrà leggere i discendenti di ogni nodo e inizia con un valore pari a 1 per il primo figlio di un nodo genitore.

Nota: *ci possono essere nodi con uguale livello e uguale ordinata, ma con genitori differenti.*

Mentre legge la direttiva OVERLAY, il linker ricorda il livello corrente e, per ogni nuovo nodo, paragona il livello specificato con questo valore. Se è minore, allora il nuovo nodo è un discendente di un nodo precedente. Se è uguale, il nuovo nodo ha lo stesso genitore di quello corrente, mentre se è maggiore il nuovo nodo è un diretto discendente di quello corrente, e quindi il nuovo livello deve essere aumentato di 1 rispetto a quello corrente.

Gli esempi riportati nella pagina successiva possono aiutare a chiarire il procedimento.

Direttiva	Livello	Ordinata	Albero
OVERLAY			ROOT
a	1	1	
b	1	2	a b c
c	1	3	
#			
OVERLAY			ROOT
a	1	1	
b	1	2	a b
*c	2	1	
*d	2	2	c d
#			
OVERLAY			ROOT
a	1	1	
b	1	2	
*c	2	1	a b e f l
*d	2	2	
e	1	3	c d g h k
f	1	4	
*g	2	1	i j
*h	2	2	
**i	3	1	
**j	3	2	
*k	2	3	
l	1	5	
#			

Tavola 4-A

I valori di livello e di ordinata forniti in precedenza si riferiscono al nodo indicato sulla stessa linea. Tutti i file dell'esempio precedente avrebbero potuto essere liste di file: le lettere singole sono state usate per chiarezza. Per esempio la Tavola 4-B

```
ROOT bin/mainaaa
OVERLAY
bin/mainbbb,bin/mainccc,bin/mainddd
*bin/makereal
*bin/trbblok,bin/transint,bin/transr,*
bin/transri
bin/outcode
#
```

Tavola 4-B

dichiara l'albero della Tavola 4-C:

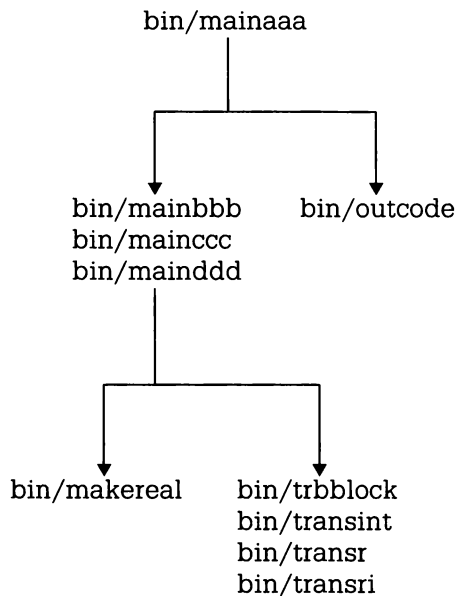



Tavola 4-C

Il linker, durante la fase di link, legge i file sovrapponibili nell'ordine che è stato specificato nella direttiva, linea dopo linea. Quest'ordine viene preservato anche nella mappa di link e nella tavola di riferimento incrociato, in modo che si possa dedurre l'esatta struttura dell'albero di overlay dal livello, e dall'ordinata indicata insieme a ogni segmento di codice.

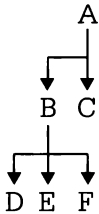
4.3.2 Riferimenti ai simboli

Il linker, mentre effettua il link di un programma con overlay, controlla la validità di tutti i riferimenti ai simboli

Supponiamo che il riferimento sia nel nodo dell'albero chiamato "R" e che il simbolo sia nel nodo "S". Il riferimento è corretto solo se una delle seguenti condizioni è vera.

(a) R e S sono lo stesso nodo.
(b) R è un discendente di S.
(c) R è il genitore di S.

I riferimenti del terzo tipo sono noti come **riferimenti in overlay**. Per risolvere i riferimenti quando il programma è in esecuzione, ci si serve del supervisore degli overlay. Il supervisore controlla se il segmento di codice contenente il simbolo richiesto è già in memoria. Se non lo è, viene rimosso il segmento posto allo stesso livello con tutti i suoi successori, e il nodo contenente il simbolo viene trasferito in memoria. Un segmento di codice sovrapposto ritorna direttamente al segmento che lo ha richiamato senza essere rimosso dalla memoria, finché non viene coperto da un altro nodo.

Per esempio, supponiamo che l'albero sia:

Quando viene caricato il programma, solo A è in memoria. Quando in A viene trovato un riferimento a un simbolo in B, viene caricato e usato B. Se a sua volta B richiama D, viene caricato un altro nodo. Quando B ha terminato e ritorna ad A, sia B che D vengono lasciati in memoria, in modo che possano essere utilizzati nuovamente senza bisogno di essere ricaricati. Supponiamo ora che A richiami C. Il supervisore rimuove i segmenti di codice che non servono, in questo caso B e D, e sui quali quindi è possibile sovrapporre il segmento richiesto. Una volta che è stata liberata la memoria, viene caricato C. Quindi, quando il linker esegue un certo nodo, sono presenti in memoria tutti i nodi precedenti fino alla radice e magari anche qualche discendente.

4.3.3 Precauzioni e avvertimenti

Il linker presume che tutti i riferimenti in overlay gestiti dal supervisore siano salti o chiamate di subroutine. Quindi non dovete usare simboli di overlay come etichette di dati.

Bisogna evitare di usare codice "sporco", perché il linker non sempre carica un nodo che è stato liberato dal file caricabile.

Il linker associa a ogni simbolo che ha un riferimento in un altro nodo un **numero di overlay**, che usa per generare l'etichetta per il supervisore delle sovrapposizioni relativa a quel simbolo. Questa etichetta è nel formato "OVLYnnnn", dove nnnn, un intero maggiore o uguale a zero, è il numero di overlay. Di conseguenza non si possono usare simboli aderenti a questo

formato in nessuna parte del proprio programma.

Il linker raggruppa tutte le sezioni di programma che hanno lo stesso nome, in modo che possano essere caricate in memoria consecutivamente.

4.4 Codici e messaggi d'errore

Questi errori dovrebbero verificarsi raramente. Di norma, comunque, la responsabilità è del compilatore e non del programma. In ogni caso bisogna controllare che sia stato mandato al linker un programma adatto (per esempio un programma di input deve avere un'unità introduttiva che indichi al linker l'arrivo di un programma).

Moduli oggetto non validi

2 Uso non valido di un simbolo in overlay
3 Uso non valido di un simbolo
4 Uso non valido di un common
5 Uso non valido di un riferimento in overlay
6 Riferimento in overlay diverso da zero
7 Rilocazione di un blocco esterno non valida
8 Rilocazione bss non valida
9 Rilocazione di un'unità di programma non valida
10 Offset scorretto durante una rilocazione a 32 bit
11 Offset scorretto durante una rilocazione a 16/8 bit
12 Offset scorretto con un riferimento a 32 bit
13 Offset scorretto con un riferimento a 16/8 bit
14 Fine del file non attesa
15 Hunk.end mancante
16 Chiusura del file non valida
17 Chiusura prematura del file
18 Chiusura prematura del file

Errori interni

19 Tipo non valido nella lista di hunk
20 Errore interno durante la scansione della libreria
21 Argomento freevector non valido
22 Simbolo non definito nella seconda fase

Appendice

Input e output da console dell'Amiga

Nota: *in questa appendice i caratteri "<CSI>" rappresentano il prefisso delle sequenze di controllo (dall'inglese Control Sequence Introducer). In output si può usare sia la sequenza di caratteri ESC-[, sia il valore di un byte $9B (esadecimale). In input si riceve $9B.*

Introduzione

Questa appendice descrive i vari modi in cui è possibile operare l'input e l'output tramite la console virtuale (tastiera e schermo) dell'Amiga. È possibile aprire questa console come un qualunque file dell'AmigaDOS (con "*", "CON:" o "RAW:") oppure effettuando chiamate dirette al console.device. Di seguito sono riportate le caratteristiche delle diverse modalità d'uso.

* "Asterisco" non apre nessuna finestra, ma usa la finestra CLI corrente. L'utente riceve le lettere minuscole a-z, quelle maiuscole A-Z, i numeri, i simboli ASCII speciali e i caratteri di controllo, mentre le sequenze di caratteri complesse non sono accettate. Fondamentalmente, si possono ricevere i caratteri che una telescrivente potrebbe generare con la pressione di un singolo tasto. Inoltre è possibile, premendo contemporaneamente il tasto ALT, ricevere ognuno di questi caratteri con il bit più significativo posto a uno ($80-$FF). La modifica della linea viene effettuata in vostra vece; questo significa che l'AmigaDOS accetta i caratteri di controllo <BACKSPACE> e CTRL-X per la cancellazione dei caratteri e delle linee. Qualunque

sequenza <CSI> viene elaborata automaticamente così come i caratteri C, D, E, F, H e X premuti con il tasto CTRL. Qualunque carattere <CR> o CTRL-M viene convertito in CTRL-J (linea nuova).

CON: Si comporta nella stessa maniera di "*" a eccezione del fatto che rende possibile definire una nuova finestra.

RAW: Nel caso più semplice RAW: non adotta, in confronto a CON:, le funzioni per la modifica della linea (perché non è in grado di interpretare i codici di controllo), ma consente il controllo dei tasti funzione e del cursore. Questi sono segnalati con una sequenza di caratteri che può essere elaborata in maniera intelligente per i propri scopi.
Nel caso più complesso si possono inviare altri comandi (per mezzo di scritture a RAW:) al gestore della console per ricevere informazioni più dettagliate. Per esempio si possono richiedere informazioni sulla pressione e il rilascio dei tasti oppure dati sugli eventi del mouse. Riferirsi a "Selezione degli eventi di input di RAW:" a pag. 277 per sapere come richiedere queste informazioni.

console.device Con questo metodo si ottiene il pieno controllo del device. È possibile cambiare la KeyMap con una di propria ideazione ed è possibile "riprogettare" completamente la tastiera.

ATTENZIONE: le informazioni riportate in questo capitolo possono non essere applicabili nello stesso modo ai modelli A500/1000/2000.

Comandi dell'AmigaDOS di utilità

Due comandi AmigaDOS molto utili permettono di sperimentare queste funzioni. Il primo:

```
TYPE RAW:10/10/100/30/ opt h
```

accetta l'input da una finestra RAW: e visualizza il risultato in esadecimale e ASCII. Questo comando fornisce un modo molto semplice per conoscere con sicurezza quali caratteri vengono mandati dalla tastiera.

Il secondo:

```
COPY ''RAW:10/10/100/30/Input RAW'' ''RAW:100/10/200/100/Output RAW''
```

permette di scrivere una sequenza nella finestra di input e di vedere i movimenti del cursore nella finestra di output. Dal momento che COPY non riconosce l'end-of-file con un input RAW:, si deve operare un reboot del sistema quando si è terminato l'uso di questo comando.

Input di tastiera CON:

L'input proveniente dalla tastiera, se viene letto per mezzo del device CON:, viene preelaborato.

La maggior parte dei programmi che fanno uso della tastiera, leggono i caratteri ASCII, come "B", dal device CON:. I programmi speciali come i word processor o quelli musicali che sfruttano a fondo la tastiera fanno invece uso del device RAW:.

Per generare i caratteri internazionali e quelli speciali con la tastiera, occorre premere il tasto ALT assieme ad altri. Questa operazione assegna valore 1 al bit più significativo del codice ASCII generato dalla pressione del secondo tasto.

La generazione di $FF (y con dieresi) è un caso particolare. Se si seguisse la convenzione standard, sarebbe generato da ALT-DEL. Dal momento che il codice ASCII <Del> ($7F esadecimale) non è un carattere stampabile, e fa parte della filosofia dei progettisti che le combinazioni di questo tipo non generino caratteri stampabili, la combinazione ALT-DEL è stata sostituita con ALT-"-" della tastierina numerica.

Nota: *questa combinazione di tasti vale esclusivamente per l'Amiga 1000.*

La Tavola A-1 contiene una lista dei caratteri visualizzabili sull'Amiga. I caratteri NBSP (NonBreak SPace) e SHY (Soft HYphen) sono usati nei word processor per specificare uno spazio e la divisione in sillabe, senza alterare le proprietà del carattere.

<table>
<tr><td></td><td></td><td></td><td></td><td>b8</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td></td><td></td><td></td><td>b7</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td></td><td></td><td></td><td>b6</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td></td><td></td><td></td><td></td><td>b5</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td></td><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>00</td><td></td><td></td><td>SP</td><td>0</td><td>@</td><td>P</td><td>`</td><td>p</td><td></td><td></td><td>NBSP</td><td>•</td><td>À</td><td>Ð</td><td>à</td><td>ð</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>01</td><td></td><td></td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td><td></td><td></td><td>¡</td><td>±</td><td>Á</td><td>Ñ</td><td>á</td><td>ñ</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>02</td><td></td><td></td><td>"</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td><td></td><td></td><td>¢</td><td>²</td><td>Â</td><td>Ò</td><td>â</td><td>ò</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>03</td><td></td><td></td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td><td></td><td></td><td>£</td><td>³</td><td>Ã</td><td>Ó</td><td>ã</td><td>ó</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>04</td><td></td><td></td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td><td></td><td></td><td>¤</td><td>´</td><td>Ä</td><td>Ô</td><td>ä</td><td>ô</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>05</td><td></td><td></td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td><td></td><td></td><td>¥</td><td>µ</td><td>Å</td><td>Õ</td><td>å</td><td>õ</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>06</td><td></td><td></td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td><td></td><td></td><td>¦</td><td>¶</td><td>Æ</td><td>Ö</td><td>æ</td><td>ö</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>07</td><td></td><td></td><td>'</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td><td></td><td></td><td>§</td><td>·</td><td>Ç</td><td>×</td><td>ç</td><td>÷</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>08</td><td></td><td></td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td><td></td><td></td><td>¨</td><td>¸</td><td>È</td><td>Ø</td><td>è</td><td>ø</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>09</td><td></td><td></td><td>)</td><td>9</td><td>I</td><td>Y</td><td>i</td><td>y</td><td></td><td></td><td>©</td><td>¹</td><td>É</td><td>Ù</td><td>é</td><td>ù</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>10</td><td></td><td></td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td><td></td><td></td><td>ª</td><td>º</td><td>Ê</td><td>Ú</td><td>ê</td><td>ú</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>11</td><td></td><td></td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td>{</td><td></td><td></td><td>«</td><td>»</td><td>Ë</td><td>Û</td><td>ë</td><td>û</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>12</td><td></td><td></td><td>,</td><td><</td><td>L</td><td>\</td><td>l</td><td>|</td><td></td><td></td><td>¬</td><td>¼</td><td>Ì</td><td>Ü</td><td>ì</td><td>ü</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>13</td><td></td><td></td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td>}</td><td></td><td></td><td>SHY</td><td>½</td><td>Í</td><td>Ý</td><td>í</td><td>ý</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>14</td><td></td><td></td><td>.</td><td>></td><td>N</td><td>^</td><td>n</td><td>~</td><td></td><td></td><td>®</td><td>¾</td><td>Î</td><td>Þ</td><td>î</td><td>þ</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>15</td><td></td><td></td><td>/</td><td>?</td><td>O</td><td>_</td><td>o</td><td></td><td></td><td></td><td>¯</td><td>¿</td><td>Ï</td><td>ß</td><td>ï</td><td>ÿ</td></tr>
</table>

Tavola A-1: codici internazionali dei caratteri

Nota: *l'AmigaDOS usa l'input derivante da CON: per il CLI e la maggior parte degli altri comandi, filtrando di conseguenza tutti gli input dei tasti funzione e di quelli per il controllo del cursore. I programmi che vengono eseguiti in ambiente AmigaDOS possono (e infatti alcuni lo fanno) comunque usare il device RAW: ed elaborare l'input dei tasti funzione.*

Nota: *"NBSP" è un sequenza nonbreak. "SHY" è un soft-hyphen.*

Output di schermo del device CON:

L'output di schermo di CON: è esattamente uguale a quello di RAW: eccetto per il fatto che il carattere <LF> ($0A esadecimale) viene trasformato in carattere di newline. L'effetto che si ottiene ogni volta che viene visualizzato un <LF> è quello di muovere il cursore all'inizio della linea successiva.

Output di schermo del device RAW:

Per scrivere un testo sul video vengono adottati i codici ANSI x3.64.

Funzioni di controllo indipendenti (senza prefisso):

Ctrl	**Hex**	**Nome**	**Definizione**	**Descrizione**
H	08	BS	Backspace	Muove il cursore a sinistra di una colonna.
I	09	TAB	Tab	Muove il cursore a destra di una colonna.
J	0A	LF	Line Feed	
K	0B	VT	Vertical Tab	Muove il cursore in su di una linea, effettuando uno scroll se necessario.
L	0C	FF	Form Feed	Pulisce lo schermo.
M	0D	CR	Carriage Return	Sposta il cursore alla prima colonna.
N	0E	SO	Shift Out	Assegna valore 1 al MSB (bit più significativo) di ogni carattere prima della visualizzazione.
O	0F	SI	Shift In	Operazione contraria a Shift Out.
[	1B	ESC	Escape	Vedere in seguito.

Precedere il carattere seguente con <ESC> per eseguire l'azione indicata.

Chr	**Nome**	**Definizione**	**Descrizione**
c	RIS	Reset To Initial State	Riporta allo stato iniziale.

Precedere i caratteri seguenti con <ESC>, o premerli simultaneamente a CTRL-ALT per eseguire le azioni indicate.

Chr	**Hex**	**Nome**	**Definizione**	**Descrizione**
D	84	IND	Index	Muove il cursore verso il basso.
E	85	NEL	Next Line	Muove il cursore all'inizio della linea successiva.
M	8D	RI	Reverse Index	Muove il cursore verso l'alto.
[	9B	CSI	Control Sequence Introducer	Indicatore dell'inizio di una sequenza di controllo. Sull'A500 e l'A2000 il <CSI> da tastiera si ottiene con la combinazione di tasti: ALT-ESC.

Analizziamo ora le sequenze di controllo (introdotte da <CSI>) con parametri. Il primo carattere nella tavola seguente (sotto la colonna <CSI>) rappresenta il numero di parametri disponibili:

"0" indica che non è permesso alcun parametro.
"1" indica 0 o 1 parametri numerici.
"2" indica 2 parametri numerici ("14;94").
"3" un qualunque numero di parametri numerici, separati dal punto e virgola (;).
"4" indica esattamente 4 parametri.
"8" indica esattamente 8 parametri.

<CSI>	**Nome**	**Definizione**	**Descrizione**
1 @	ICH	Insert Character	Inserisce uno o più spazi, spostando il resto della linea verso destra.
1 A	CUU	Cursor Up	Muove il cursore in su di una linea.
1 B	CUD	Cursor Down	Muove il cursore in giù di una linea.
1 C	CUF	Cursor Forward	Muove il cursore in avanti di un carattere.
1 D	CUB	Cursor Backward	Muove il cursore indietro di un carattere.
1 E	CNL	Cursor Next Line	Muove il cursore in giù di n linee e alla colonna 1.
1 F	CPL	Cursor Preceding Line	Muove il cursore in su di n linee e alla colonna 1.
2 H	CUP	Cursor Position	"<CSI>riga;colonnaH"
1 J	ED	Erase In Display	(Cancella solo fino alla fine dello schermo).
1 K	EL	Erase In Line	(Cancella solo fino alla fine della linea).
1 L	IL	Insert Line	Inserisce una linea vuota prima di quella contenente il cursore.
1 M	DL	Delete Line	Cancella la linea corrente, muove le linee seguenti in su di una e lascia l'ultima vuota.
1 P	DCH	Delete Character	Cancella un carattere.
2 R	CPR	Cursor Position Report	(Solamente nel canale di lettura). Formato: "<CSI>riga;colonnaR".
1 S	SU	Scroll Up	Rimuove una linea dal-

			l'inizio dello schermo, muove le altre in su di una e lascia l'ultima vuota.
1 T	SD	Scroll Down	Rimuove una linea dalla fine dello schermo, muove le altre in giù di una e lascia la prima vuota.
3 h	SM	Set Mode	<CSI>20h obbliga RAW: a convertire durante l'output <LF> in <newline>.
3 l	RM	Reset Mode	<CSI>20l agisce in modo contrario a SET MODE 20.
3 m	SGR	Select Graphic Rendition	Seleziona la traduzione grafica.
1 n	DSR	Device Status Report	Riporta lo stato del dispositivo.

Le sequenze che seguono non sono standard ANSI, ma sono sequenze caratteristiche dell'Amiga.

<CSI>	**Nome**	**Definizione**	**Descrizione**
1 t	aSPL	Set Page Lenght	Imposta la lunghezza di pagina.
1 u	aSLL	Set Line Lenght	Imposta la lunghezza di linea.
1 x	aSLO	Set Left Offset	Imposta l'offset sinistro.
1 y	aSTO	Set Top Offset	Imposta l'offset superiore.
3 {	aSRE	Set Raw Events	Imposta gli eventi RAW:.
8 |	aIER	Input Event Report	Input informazioni sull'evento (lettura).
3 }	aRRE	Reset Raw Events	Disattiva gli eventi RAW:.
1 ~	aSKR	Special Key Report	Informazioni speciali sul carattere (lettura).
1 p	aSCR	Set Cursor Rendition	Attiva la visualizzazione del cursore, <ESC> p spegne il cursore.
0 q	aWSR	Window Status Request	Richiesta dello stato della finestra.
4 r	aWBR	Window Bounds Report	Informazioni sui limiti della finestra (lettura).

Esempi:

Per muovere il cursore a destra di 1:

`<CSI>C` oppure `<Tab>` oppure `<CSI>1C`

Per muovere il cursore a destra di 20:

`<CSI>20C`

Per muovere il cursore nell'angolo superiore sinistro (home):

`<CSI>H` oppure `<CSI>1;1H` oppure `<CSI>;1H` oppure `<CSI>1;H`

Per muovere il cursore alla quarta colonna della prima riga della finestra:

`<CSI>1;4H` oppure `<CSI>;4H`

Per pulire lo schermo:

`<FF>` oppure `CTRL-L`	(carattere di pulizia schermo)
`<CSI>H<CSI>J`	(home e pulisce fino alla fine dello schermo)
`<CSI>H<CSI>23M`	(home e cancella 23 linee)
`<CSI>H<CSI>23L`	(home e inserisce 23 linee)

Input da tastiera RAW:

La lettura dell'input dal device RAW: restituisce una serie di byte standard ANSI x3.64. Questo flusso di dati può contenere caratteri normali, come informazioni riguardanti gli eventi di input di RAW:. È possibile inoltre richiedere altri eventi di input usando le sequenze di controllo SET RAW EVENTS (aSRE) e RESET RAW EVENTS (aRRE); per i particolari riferirsi a "Selezione degli eventi di input di RAW:" nella pagina successiva.

Se si effettua una richiesta di input a RAW: e non esiste alcun carattere in attesa di venir prelevato (carattere pendente), il comando di lettura attende finché non giunge qualche input. È possibile verificare la presenza di caratteri pendenti operando una richiesta "WaitForChar".

Nello stato di default i tasti di funzione e del cursore inviano al processo le sequenze seguenti:

Tasto	**Senza shift**	**Con shift**
F1	<CSI>0~	<CSI>10~
F2	<CSI>1~	<CSI>11~
F3	<CSI>2~	<CSI>12~
F4	<CSI>3~	<CSI>13~
F5	<CSI>4~	<CSI>14~

F6	<CSI>5~	<CSI>15~
F7	<CSI>6~	<CSI>16~
F8	<CSI>7~	<CSI>17~
F9	<CSI>8~	<CSI>18~
F10	<CSI>9~	<CSI>19~
HELP	<CSI>?~	<CSI>?~ (identico)
Tasti cursore:		
Su	<CSI>A	<CSI>T
Giù	<CSI>B	<CSI>S
Destra	<CSI>C	<CSI> A (notare lo spazio)
Sinistra	<CSI>D	<CSI> @ (notare lo spazio)

Selezione degli eventi di input di RAW:

Se si sta usando RAW: per default, si ottengono i dati ANSI e le sequenze di controllo menzionate in precedenza. Se queste informazioni non sono sufficienti se ne possono richiedere altre dal driver della console.

Se, per esempio, si vuole sapere quando ogni tasto viene premuto e rilasciato, occorre richiedere un "RAW keyboard input". Questo si realizza mandando "<CSI>1{" alla console. Di seguito è riportata una lista delle richieste di input valide.

Tipi di eventi di input di RAW:

Numero di richiesta	**Descrizione**	**Nota**
0	nop (nessuna operazione)	Usato internamente.
1	Input da tastiera RAW	
2	Input da mouse RAW	
3	Finestra	Viene inviato ogni volta che la finestra diventa attiva.
4	Posizione del puntatore	
5		Non usato.
6	Timer	
7	Gadget selezionato	
8	Gadget rilasciato	
9	Requester in attività	
10	Numero di menu	
11	Gadget chiusura finestra attivato	

12	Alterazione delle dimensioni della finestra.
13	Aggiornamento della finestra.
14	Cambiamento delle preferenze.
15	Disco rimosso.
16	Disco inserito.

Se scegliete uno di questi eventi, comincerete a ottenere informazioni su di esso nella forma che segue:

```
<CSI><classe>
<sottoclasse>
<keycode>
<qualificatori>
<x>
<y>
<secondi>
<microsecondi>
```

<CSI> è un campo composto da un byte che contiene il Control Sequence Introducer, ovvero $9B in esadecimale.

<classe> è il tipo di evento di input di RAW, come da tavola precedente.

<sottoclasse> non è usato attualmente ed è sempre zero.

<keycode> indica quale tasto è stato premuto (vedere Tavola A-2 e Tavola A-3). Questo campo viene usato anche per informazioni sul mouse.

Il campo <qualificatori> indica lo stato della tastiera e del sistema. I qualificatori sono definiti come segue:

Bit	**Maschera**	**Tasto**	
0	0001	shift sinistro	
1	0002	shift destro	
2	0004	caps lock	* speciale, vedere più avanti.
3	0008	control	
4	0010	alt sinistro	
5	0020	alt destro	
6	0040	pressione tasto Amiga sinistro	
7	0080	pressione tasto Amiga destro	
8	0100	tastierina numerica	
9	0200	repeat	
10	0400	interrupt	Attualmente non viene usato.
11	0800	multi broadcast	La finestra corrente oppure tutte le finestre.

12	1000	pulsante sinistro del mouse	
13	2000	pulsante destro del mouse	
14	4000	pulsante centrale del mouse	(Non disponibile sul mouse standard).
15	8000	mouse relativo	Indica che le coordinate del mouse sono relative e non assolute.

Il tasto CAPS LOCK è gestito in una maniera particolare: genera un codice solamente quando viene premuto e non quando viene rilasciato. Comunque il bit di premuto e non premuto ($80 esadecimale) viene riportato. Se la pressione del tasto CAPS LOCK causa l'accensione del LED, viene inviato il codice 62 (CAPS LOCK premuto). Se invece la pressione del tasto causa lo spegnimento del LED, viene generato il codice 190 (CAPS LOCK rilasciato). In effetti la tastiera segnala questo tasto come premuto finché non lo si preme una seconda volta.

Con l'input di tastiera RAW:, i tasti selezionati non generano un semplice carattere da "A" a "Z" ma restituiscono un "raw keycode report" avente il seguente formato:

```
<CSI>1;0;<keycode>;<qualificatori>;0;0;<secondi>;<microsecondi>|
```

I campi <secondi> e <microsecondi> contengono il valore del tempo di sistema quando si è verificato l'evento. Questi valori sono memorizzati in long word e come tali possono raggiungere teoricamente i quattro miliardi. Per esempio, se l'utente premesse e rilasciasse il tasto "B" con lo SHIFT sinistro e il tasto Amiga destro premuti, si riceverebbero dati sul tipo dei seguenti:

```
<CSI>1;0;35;129;0;0;23987;99|
<CSI>1;0;163;129;0;0;24003;18|
```

I campi "0;0" non sono usati dalla tastiera ma vengono utilizzati quando si seleziona l'input del mouse. Questi campi, con l'input del mouse, indicano rispettivamente le coordinate X e Y del mouse.

Il campo <keycode> contiene un valore decimale che rappresenta il tasto premuto o rilasciato. L'aggiunta di 128 al codice della pressione del tasto dà origine al codice del rilascio. La Tavola A-2 permette di convertire velocemente da un tasto al suo keycode. La Tavola A-3 permette di convertire velocemente da keycode a tasto.

Tavola A-2: copia ridotta dell'assegnazione della tastiera dell'A1000

I valori di default forniti nella tavola seguente corrispondono a:

1) I valori che il device CON: restituisce quando questi tasti sono premuti.
2) Le scritte sui tasti così come sono fornite sulla tastiera americana standard.

Tavola A-3: conversione da keycode a tasto

Numero del tasto raw	**Valore di default senza shift**	**Valore di default con shift**
00	` (accento grave)	~(tilde)
01	1	!
02	2	@
03	3	#
04	4	$
05	5	%
06	6	^
07	7	&
08	8	*
09	9	(
0A	0	)
0B	- (meno)	_ (underscore)
0C	=	+
0D	\	|
0E	non definito	
0F	0	0 (tastierina numerica)
10	Q	q
11	W	w
12	E	e
13	R	r
14	T	t
15	Y	y
16	U	u
17	I	i
18	O	o
19	P	p
1A	{	[
1B	}	]
1C	non definito	
1D	1	1 (tastierina numerica)
1E	2	2 (tastierina numerica)
1F	3	3 (tastierina numerica)
20	A	a
21	S	s
22	D	d
23	F	f
24	G	g

Numero del tasto raw	**Valore di default senza shift**	**Valore di default con shift**
25	H	h
26	J	j
27	K	k
28	L	l
29	:	;
2A	"	' (apostrofo)
2B	(Riservato)	(Riservato)
2C	non definito	
2D	4	4 (tastierina numerica)
2E	5	5 (tastierina numerica)
2F	6	6 (tastierina numerica)
30	(Riservato)	(Riservato)
31	Z	z
32	X	x
33	C	c
34	V	v
35	B	b
36	N	n
37	M	m
38	<	, (virgola)
39	>	. (punto)
3A	?	/
3B	non definito	
3C	.	. (tastierina numerica)
3D	7	7 (tastierina numerica)
3E	8	8 (tastierina numerica)
3F	9	9 (tastierina numerica)
40	Spazio	
41	Backspace	
42	Tab	
43	Enter	Enter (tastierina numerica)
44	Return	
45	Escape	<ESC>
46	Del	
47	non definito	
48	non definito	
49	non definito	
4A	-	- (tastierina numerica)
4B	non definito	

Numero del tasto raw	**Valore di default senza shift**	**Valore di default con shift**
4C	Cursore in su.	Scroll in giù
4D	Cursore in giù.	Scroll in su
4E	Cursore in avanti.	Scroll a sinistra
4F	Cursore indietro.	Scroll a destra
50	F1	<CSI>10~
51	F2	<CSI>11~
52	F3	<CSI>12~
53	F4	<CSI>13~
54	F5	<CSI>14~
55	F6	<CSI>15~
56	F7	<CSI>16~
57	F8	<CSI>17~
58	F9	<CSI>18~
59	F10	<CSI>19~
5A	non definito.	
5B	non definito.	
5C	non definito.	
5D	non definito.	
5E	non definito.	
5F	Help.	
60	Shift (a sinistra della barra spaziatrice).	
61	Shift (a destra della barra spaziatrice).	
62	Caps lock.	
63	Control.	
64	Alt sinistro.	
65	Alt destro.	
66	"Amiga" (a sinistra della barra spaziatrice).	
67	"Amiga" (a destra della barra spaziatrice).	

Numero del tasto raw	Valore di default senza shift	Valore di default con shift
68	Pulsante sinistro del mouse (non convertito). **Nota:** *gli input riguardanti il mouse si riferiscono solo al mouse connesso a Intuition tramite la porta 1.*	
69	Pulsante destro del mouse (non convertito).	
6A	Pulsante centrale del mouse (non convertito).	
6B	non definito.	
6C	non definito.	
6D	non definito.	
6E	non definito.	
6F	non definito.	
70-7F	non definiti.	
80-F8	transizione verso l'alto (rilascio di un tasto) di uno dei tasti precedenti. 80 sta per 00 e F8 per 78.	
F9	L'ultimo keycode non era valido (era stato mandato per ri-sincronizzare).	
FA	Overflow del buffer di tastiera.	
FB	Non definito; riservato per il malfunzionamento del processore della tastiera.	
FC	L'auto-test della tastiera è fallito.	

Numero del tasto raw	**Valore di default senza shift**	**Valore di default con shift**
FD	Inizio del flusso di tasti premuti all'accensione. I tasti premuti durante il processo di inizializzazione sono inviati tra FD e FE.	
FE	Termine del flusso di tasti premuti all'accensione.	
FF	Evento del mouse, solamente movimento. Nessun cambiamento dei pulsanti (non convertito).	

Note riguardo la tavola precedente:

1) "non definito" indica che la tastiera corrente non genera questo numero. Se si sta usando "SetKeyMap" per modificare la mappa della tastiera, devono essere incluse anche le voci relative a questi numeri.
2) "(non convertito)" si riferisce agli eventi dei pulsanti del mouse. È possibile usare la sequenza "<CSI>2{" per informare il driver della console che si desidera ricevere gli eventi del mouse; altrimenti questi ultimi non vengono trasmessi.
3) "(Riservato)" indica che il keycode è stato riservato per le tastiere non americane. Il tasto corrispondente al codice "2B" si trova tra le virgolette e il tasto di RETURN. Il codice "30" tra lo SHIFT e il tasto "Z".

Il Manuale dell'AmigaDOS di riferimento tecnico

Indice

1. Il filing system 289
2. Struttura dei file binari dell'Amiga 298
3. Strutture dati dell'AmigaDOS 318
4. Informazioni aggiuntive sull'AmigaDOS
per la programmazione avanzata 337

Capitolo 1

Il filing system

Questo capitolo descrive il filing system dell'AmigaDOS e contiene le informazioni per il recupero di un disco rovinato da errori hardware.

1.1 Struttura dei file dell'AmigaDOS
1.1.1 Root block
1.1.2 User Directory block
1.1.3 File Header block
1.1.4 File List block
1.1.5 Data block
1.2 DISKED - L'editor per i dischi

1.1 Struttura dei file dell'AmigaDOS

Il gestore dei file dell'AmigaDOS impiega dischi formattati in blocchi di dimensioni uguali ed è in grado di gestire una struttura gerarchica di directory con profondità non definita, nella quale ogni directory può contenere altre directory e file oppure solamente file. La struttura è un albero puro, e quindi non consente la creazione di riferimenti a ritroso.

La struttura dei dati su disco è tale da rendere possibile il recupero di quasi tutto, se non tutto, il contenuto di un disco afflitto da un grave errore hardware. Per ricostruire il contenuto di un disco si usa il comando DISKED, che viene illustrato, insieme alla sua sintassi, nella sezione 1.2, "DISKED - L'editor per i dischi". Prima di poter ricostruire il contenuto di un disco, bisogna comprenderne la struttura. I paragrafi seguenti descrivono il formato dei vari blocchi.

1.1.1 Root block

La radice dell'albero è il Root block, che ha una collocazione fissa nel disco. La radice è come ogni altra directory, eccetto per il fatto che non ha genitore e per il tipo secondario che è differente. L'AmigaDOS memorizza il nome di volume del disco nel campo del nome del Root block.

Ogni blocco del filing system contiene un valore appropriato di checksum, che fa in modo che la somma di tutte le word del blocco (escludendo i riporti) sia zero.

La tavola seguente descrive il formato del Root block.

Offset	Campo	Descrizione
0	T.SHORT	Tipo
1	0	Chiave d'intestazione (sempre 0)
2	0	Numero sequenziale più alto (sempre 0)
3	HT SIZE	Dimensione della tavola di hash (uguale alla dimensione del blocco meno 56)
4	0	
5	CHECKSUM	
6	tavola degli hash	
SIZE-51		
SIZE-50	BMFLAG	TRUE (vero) se la bitmap del disco è valida
SIZE-49	pagine della bitmap	Usato per indicare i blocchi contenenti la bitmap
SIZE-24		
SIZE-23	DAYS	Data e ora dell'ultima modifica del volume
SIZE-22	MINS	
SIZE-21	TICKS	
SIZE-20	DISK NAME	Nome del volume sotto forma di stringa BCPL composta da meno di 31 caratteri
SIZE-7	CREATEDAYS	Data e ora di creazione del volume
SIZE-6	CREATEMINS	
SIZE-5	CREATETICKS	
SIZE-4	0	Voce successiva nella catena di hash (sempre 0)
SIZE-3	0	Directory genitore (sempre 0)
SIZE-2	0	Estensione (sempre 0)
SIZE-1	ST.ROOT	Il tipo secondario indica il blocco radice

Tavola 1-A: Root block

1.1.2 User Directory block

La tavola seguente descrive il formato di un User Directory block.

Offset	Campo	Descrizione
0	T.SHORT	Tipo
1	OWN KEY	Chiave d'intestazione (puntatore a se stesso)
2	0	Numero sequenziale più alto (sempre 0)
3	0	
4	0	
5	CHECKSUM	
6 … SIZE-51	tavola degli hash	
SIZE-50	inutilizzato	
SIZE-48	PROTECT	Bit di protezione
SIZE-47	0	Non usato (sempre 0)
SIZE-46 … SIZE-24	COMMENT	Commento memorizzato come stringa BCPL
SIZE-23	DAYS	Data e ora della creazione
SIZE-22	MINS	
SIZE-21	TICKS	
SIZE-20	DIRECTORY NAME	Nome della directory sotto forma di stringa BCPL composta da meno di 31 caratteri
SIZE-4	HASHCHAIN	Voce successiva con lo stesso valore di hash
SIZE-3	PARENT	Puntatore alla directory genitore
SIZE-2	0	Estensione (sempre 0)
SIZE-1	ST.USERDIR	Tipo secondario

Tavola 1-B: User Directory block

I blocchi delle directory utente hanno un tipo T.SHORT e un tipo secondario ST.USERDIRECTORY. Le sei word di informazioni all'inizio del blocco indicano anche la chiave di proprietà del blocco (cioè, il numero del blocco), il checksum e la dimensione della tavola di hash. Questa tavola, per mezzo di particolari algoritmi, permette di individuare univocamente l'indirizzo logico del primo blocco appartenente all'oggetto cercato (directory o file). Questo metodo di ricerca, per quanto elaborato, permette di ottenere un'elevata flessibilità e velocità. Le 50 word di informazioni alla fine del blocco contengono la data e l'ora di creazione, il nome della directory, un puntatore al successivo file o directory nella catena di hash e un puntatore alla directory di livello precedente.

Per trovare un file o una directory si deve prima applicare una funzione hash al suo nome, ottenendo un offset per la tavola di hash. Questa informazione è la chiave del primo blocco della lista che unisce quelli dotati dello stesso valore di hash (o zero se non ce ne sono). L'AmigaDOS legge il blocco dotato di quella chiave e ne confronta il nome con quello cercato: se i nomi non corrispondono legge il blocco successivo e così via.

1.1.3 File Header block

La tavola seguente descrive il formato del blocco di intestazione di un file.

Tavola 1-C: File Header block

Ogni file inizia con un file header block (blocco d'intestazione del file) che è dotato di un tipo T.SHORT e di un tipo secondario ST.FILE. L'inizio e la fine del blocco contengono il nome, l'ora e le informazioni supplementari simili a quelle che si trovano in un blocco di directory. Il corpo del file consiste in una serie di indirizzi appartenenti ai diversi blocchi di dati di cui è composto; gli indirizzi dei blocchi vengono numerati in ordine crescente dal numero 1. L'AmigaDOS memorizza gli indirizzi di questi blocchi in word consecutive a partire dall'offset size-51 del blocco verso il basso. In genere l'AmigaDOS non utilizza tutto lo spazio a disposizione di questa lista e l'ultimo blocco di dati di ogni file non è necessariamente pieno.

1.1.4 File List block

Se in un file ci sono più blocchi di quelli che possono essere indicati nella lista dei blocchi, il campo EXTENSION non è zero e punta a un altro blocco che contiene l'estensione della lista degli indirizzi ai blocchi del file. La tavola seguente mostra il formato del blocco d'estensione di questa lista.

0	T.LIST	Tipo
1	OWN KEY	Chiave d'intestazione
2	BLOCK COUNT	Uguale al numero di blocchi di dati nella lista
3	DATA SIZE	Uguale al precedente
4	FIRST DATA	Primo blocco dati
5	CHECKSUM	
6		
	BLOCK N+3	
	BLOCK N+2	Lista estesa delle chiavi dei blocchi di dati
SIZE-51	BLOCK N+1	
SIZE-50	informazioni	(non usate)
SIZE-4	0	Voce successiva nella lista hash (sempre 0)
SIZE-3	PARENT	Blocco di intestazione di questo file
SIZE-2	EXTENSION	Successivo blocco di estensione
SIZE-1	ST.FILE	Tipo secondario

Tavola 1-D: File List block

Per costituire un file si usano tanti blocchi di estensione del file quanti ne sono necessari. La struttura di questo blocco è molto simile a quella di un File Header block, eccetto per il fatto che il tipo è differente e non vengono usati i campi del nome e della data.

1.1.5 Data block

La tavola seguente illustra il formato di un blocco di dati.

Tavola 1-E: Data block

In un blocco di dati solo sei word dell'intero quantitativo sono allocate per contenere informazioni utili al filing system. I contenuti di queste sei word sono:

- tipo (T.DATA)
- puntatore al File Header block
- numero d'ordine del blocco di dati
- numero delle word di dati
- puntatore al successivo Data block
- checksum

Normalmente tutti i blocchi di dati sono pieni (cioè hanno il campo DATA SIZE uguale alla dimensione del blocco meno sei) a eccezione dell'ultimo. L'ultimo blocco ha il puntatore al successivo Data block uguale a zero, e indica così la fine del file.

1.2 DISKED - L'editor per i dischi

Per ispezionare o ricostruire i blocchi di un disco si può usare l'editor per i dischi dell'AmigaDOS, DISKED. Questa applicazione può essere usata con ottimi risultati per recuperare informazioni da un disco rovinato, prestando però molta attenzione dal momento che DISKED scrive direttamente su disco e il rischio di produrre alterazioni accidentali non è da sottovalutare.

Si deve usare DISKED solo riferendosi al formato di un disco AmigaDOS (per una descrizione del formato, vedere le sezioni da 1.1.1 a 1.1.5 nella prima parte di questo capitolo). DISKED conosce questa struttura; per esempio il comando R (Root block) mostra la chiave del blocco contenente la radice. Con il comando G (Get block) seguito da questa chiave si copia il contenuto del blocco nella memoria e con il comando I (Info, Informazioni) vengono mostrate le informazioni contenute nelle prime e ultime locazioni, cioè il tipo del blocco, il nome, il concatenamento di hash e così via. Indicando un nome dopo il comando H (Hash), DISKED fornisce l'offset relativo a una pagina di directory per il nome che è stato fornito. Se si introduce il numero che si ottiene seguito da una barra (/), viene mostrata la chiave del blocco contenente l'intestazione. Infine si può leggerne il contenuto per mezzo di successivi comandi G.

Ipotizziamo di dover cancellare un file che, a causa di errori hardware, rende impossibile il compimento del processo di restart del filing system. Innanzitutto bisogna individuare il blocco della directory che contiene il riferimento al file in oggetto. Per far questo, occorre esaminare la struttura gerarchica delle directory, partendo dalla radice, tramite i codici hash. Successivamente si deve localizzare il riferimento al file che contiene la chiave dell'header block del file (quest'ultimo può essere sia un blocco della directory, sia un blocco d'intersezione lungo la stessa lista di hash). Per annullare il riferimento si deve introdurre il valore dell'offset, seguito da una barra (/) e da uno zero (cioè <offset>/0). Occorre poi correggere il checksum per mezzo del comando K e disabilitare la protezione in scrittura con X. Per ultimo si deve riscrivere su disco il blocco corretto con P o W. Non c'è bisogno di fare altro, visto che i blocchi del file che causano l'errore diventano nuovamente disponibili una volta che il processo di restart è riuscito a verificare l'intero stato del disco.

I comandi di DISKED sono tutti caratteri singoli e qualche volta sono seguiti da argomenti.

DISKED, nella versione 1.2, lavora con ogni tipo di disco (compresi i dischi rigidi, le partizioni e i dischi da 5,25") che sia stato installato con il comando MOUNT.

DISKED non è presente nel disco Workbench correntemente distribuito e viene incluso soltanto in alcuni pacchetti applicativi per programmatori.

Nella pagina successiva è riportato un elenco completo dei comandi disponibili.

Comando	**Funzione**
B n	Imposta a n la base logica dei numeri dei blocchi.
C n	Visualizza n caratteri a partire dall'offset corrente.
G [n]	Preleva il blocco n dal disco (il default è il numero del blocco corrente).
H nome	Calcola il valore hash del nome.
I	Visualizza le informazioni del blocco.
K	Controlla il checksum del blocco e lo corregge se è sbagliato.
L[lwp upb]	Localizza le word che corrispondono al valore della maschera (lwb e upb restringono la ricerca indicando i blocchi limite ai quali estenderla).
M n	Imposta la maschera, per i comandi L e N, a n.
N[lwp upb]	Localizza le word che non corrispondono al valore della maschera.
P n	Scrive su disco il blocco contenuto in memoria al blocco n (il default è il numero del blocco corrente).
R	Mostra il numero di blocco del Root block.
Q	Esce dal programma.
S c	Imposta il modo di visualizzazione c = C - caratteri S - stringa O - ottale X - esadecimale D - decimale.
T lwp upb	Scrive un intervallo di offset nel blocco.
V n	Imposta il valore per i comandi L e N.
W	Equivale a usare i comandi P e Q di seguito.
X	Inverte lo stato della protezione in scrittura.
Y n	Imposta la base del cilindro a n.
Z	Azzera tutte le word del buffer.
numero	Imposta la word corrente di offset nel blocco.
=	Visualizza i valori impostati nel programma.
/[n]	Visualizza la word all'offset corrente o ne aggiorna a n il valore.
'caratteri'	Scrive i caratteri all'offset corrente.
"caratteri"	Scrive una stringa all'offset corrente.

Tavola 1-F: comandi di DISKED

Per indicare un valore ottale o esadecimale bisogna anteporre al numero rispettivamente i caratteri # o #X. Si possono anche includere nelle stringhe di caratteri gli escape delle stringhe di tipo BCPL (*N e così via).

Capitolo 2

Struttura dei file binari dell'Amiga

2.1 Introduzione
2.1.1 Terminologia
2.2 Struttura dei file oggetto
2.2.1 hunk_unit
2.2.2 hunk_name
2.2.3 hunk_code
2.2.4 hunk_data
2.2.5 hunk_bss
2.2.6 hunk_reloc32
2.2.7 hunk_reloc16
2.2.8 hunk_reloc8
2.2.9 hunk_ext
2.2.10 hunk_symbol
2.2.11 hunk_debug
2.2.12 hunk_end
2.3 File caricabili
2.3.1 hunk_header
2.3.2 hunk_overlay
2.3.3 hunk_break
2.4 Esempi

2.1 Introduzione

Questo capitolo descrive in dettaglio la struttura dei file oggetto prodotti dai compilatori e dagli assemblatori. Descrive inoltre il formato – che supporta anche gli overlay – dei file binari caricabili generati dal linker e

trasferiti in memoria dal loader. Oltre a descrivere il formato dei file caricabili, questo capitolo illustra l'uso dei simboli comuni, dei riferimenti esterni assoluti e delle unità di programma.

2.1.1 Terminologia

Qui di seguito sono riportate le spiegazioni di alcuni termini tecnici adottati in questo capitolo.

Riferimenti esterni
Si può usare un nome per specificare un riferimento tra unità di programma diverse. La struttura dati permetterebbe di avere nomi più lunghi di 16 MB, ma il linker restringe la lunghezza a un massimo di 255 caratteri. Durante la fase di link, che genera un solo file caricabile da più file oggetto, bisogna assicurarsi che tutti i riferimenti esterni corrispondano a definizioni esterne. I riferimenti esterni possono essere a un byte, a una word o a una long word; le definizioni esterne si riferiscono a valori rilocabili, valori assoluti oppure librerie residenti. I riferimenti a byte o word rilocabili si riferiscono ai modi di indirizzamento relativi al PC (Program Counter) e sono interamente gestiti dal linker. La rilocazione dei riferimenti a long word può comunque avvenire durante la fase di caricamento in memoria del programma per l'esecuzione.
Occorre notare che le dimensioni dei dati in oggetto si riferiscono alla lunghezza del campo di rilocazione; è possibile, per esempio, caricare una word da un indirizzo esterno long senza che il linker controlli se l'uso del riferimento e della definizione esterna è corretto.

File oggetto
Il compilatore o l'assemblatore produce un'immagine binaria chiamata file oggetto. Un file oggetto contiene una o più unità di programma e può inoltre contenere riferimenti esterni ad altri file oggetto.

File caricabile
Il linker genera un'immagine binaria, chiamata file caricabile, da un certo numero di file oggetto. Un file caricabile non contiene nessun riferimento esterno non risolto.

Unità di programma
L'unità di programma è il più piccolo elemento trattabile dal linker. Un'unità di programma può contenere uno o più hunk; i file oggetto possono contenere una o più unità di programma. Se il linker trova un appropriato riferimento esterno all'interno di un'unità di programma mentre ispeziona una libreria associata, include l'intera unità di programma nel file caricabile. Un assemblatore produce in genere una singola unità di programma

(contenente uno o più hunk) da un solo assemblaggio; un compilatore, per esempio FORTRAN, produce un'unità di programma per ogni subroutine, per il programma principale e per i blocchi di dati. La numerazione degli hunk parte da zero per ogni unità di programma; l'unico modo con il quale ci si può riferire ad altre unità di programma è attraverso i riferimenti esterni.

Hunk
Un hunk è costituito da un blocco di codici o di dati, informazioni di rilocazione e da una lista di definizioni o riferimenti di simboli esterni. Gli hunk di dati possono specificare dati inizializzati o non inizializzati (BSS). Gli hunk BSS possono contenere definizioni esterne ma non riferimenti esterni e nemmeno valori che richiedono rilocazione. Se si pone in un overlay un blocco di dati inizializzati, il linker non dovrebbe alterarlo dal momento che lo ricarica da disco durante il processo di overlay. Gli hunk, che possono essere dotati di un nome oppure no, possono contenere una tavola di simboli atta a fornire le informazioni per il debugging simbolico. Possono inoltre contenere ulteriori informazioni di debugging da usarsi con accessori di debugging per linguaggi ad alto livello. Ogni hunk, all'interno di un'unità di programma, è dotato di un numero che parte da zero.

Libreria residente
Le librerie sono collezioni di routine di sistema; possono essere residenti in memoria, o in alternativa il sistema operativo le può caricare con una chiamata a Open Library. È possibile riferirsi alle librerie residenti attraverso i riferimenti esterni; le definizioni sono in un hunk che non contiene codice, ma soltanto una lista delle definizioni della libreria residente. Solitamente, per generare questi hunk, si assembla un file che non contiene altro che definizioni assolute esterne, per poi passare il risultato attraverso un programma speciale che converte le definizioni assolute in definizioni della libreria residente. Il linker usa il nome dell'hunk come nome della libreria residente e lo colloca nel file caricabile in modo tale che il loader possa aprire la libreria residente prima di usarla.

Libreria associata
Una libreria associata è costituita da alcuni file oggetto che contengono unità di programma le quali vengono caricate soltanto se esiste un riferimento esterno a esse. Si possono usare i file oggetto come librerie e fornirli come input primario al linker, nel qual caso vengono incluse tutte le unità di programma che i file oggetto contengono. È da notare che i file oggetto si possono concatenare.

Nodo
Un nodo contiene almeno un hunk. Un file caricabile con struttura overlay è composto da un nodo radice, residente in memoria per tutto il tempo che

il programma è in esecuzione, e da un certo numero di nodi sovrapponibili (overlay) che vengono trasferiti in memoria quando è necessario.

2.2 Struttura dei file oggetto

Il file oggetto è l'output di un assemblatore o di un compilatore. Per poter usare un file oggetto occorre risolvere tutti i riferimenti esterni; questa operazione si esegue facendo passare il file oggetto attraverso il linker. Un file oggetto è costituito da una o più unità di programma. Ogni unità di programma inizia con un'intestazione seguita da una serie di hunk uno dietro l'altro, costituiti da un certo numero di "blocchi" di vario tipo. Ogni blocco inizia con una long word che ne determina il tipo, seguita da eventuali long word aggiuntive, ed è sempre allineato secondo una long word. L'intestazione di un'unità di programma è essa stessa un blocco con questo formato.

Il formato di un'unità di programma è il seguente:

- Blocco di intestazione dell'unità di programma
- Hunk

Il formato fondamentale di un hunk è il seguente:

- Blocco del nome dell'hunk
- Blocco rilocabile
- Blocco con le informazioni di rilocazione
- Blocco con le informazioni sui simboli esterni
- Blocco della tavola dei simboli
- Blocco di debug
- Blocco finale

Si possono omettere tutti i blocchi tranne quello finale.

I paragrafi seguenti descrivono il formato di ogni blocco. Il valore della word di ciascuno appare sia in decimale sia in esadecimale dopo il nome: per esempio hunk_unit ha il valore 999 in decimale e 3E7 in esadecimale.

2.2.1 hunk_unit (999/3E7)

Specifica l'inizio di un'unità di programma. Consiste in una word contenente il tipo seguita dalla lunghezza, in long word, del nome dell'unità, seguita a sua volta dal nome stesso riempito eventualmente di zeri per essere allineato a una long word. In forma grafica il formato è quello riportato nella tavola della pagina successiva.

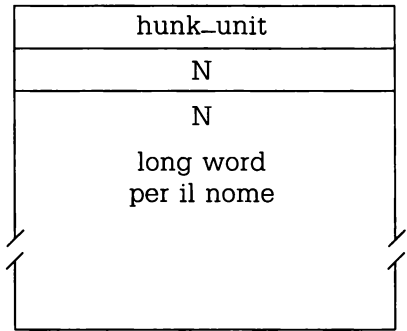



Tavola 2-A: hunk_unit (999/3E7)

2.2.2 hunk_name (1000/3E8)

Definisce il nome di un hunk. I nomi sono opzionali; se il linker trova due o più hunk che possiedono lo stesso nome, li combina insieme in un unico hunk. Occorre notare che i riferimenti esterni a 8 o 16 bit relativi al PC possono essere risolti solo tra hunk con lo stesso nome. I riferimenti esterni in un file in formato caricabile sono tra hunk differenti e richiedono un riferimento rilocabile a 32 bit; questa esigenza nasce dal fatto che, mentre il loader carica in memoria i vari hunk allocandoli nelle aree di memoria disponibili, non è possibile essere sicuri che questi siano stati allocati nella stessa area da 32K. Notate che la lunghezza è espressa in long word e che il nome del blocco, come tutti gli altri blocchi, è allineato a una long word. Il formato è quello riportato nella seguente tavola.

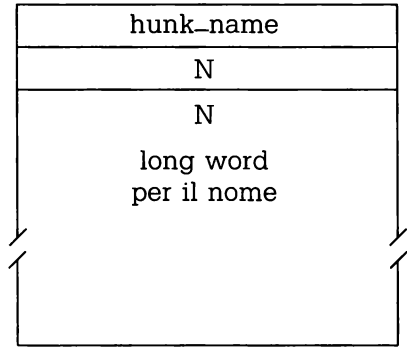



Tavola 2-B: hunk_name (1000/3E8)

2.2.3 hunk_code (1001/3E9)

Definisce un blocco di codice che deve essere caricato in memoria e possibilmente rilocato. La tavola che segue ne riporta il formato.

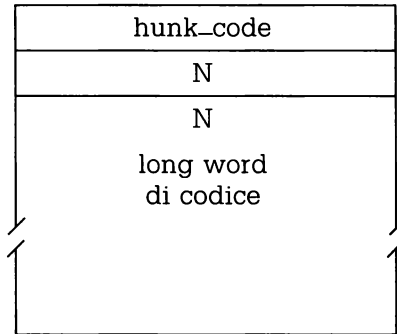



Tavola 2-C: hunk_code (1001/3E9)

2.2.4 hunk_data (1002/3EA)

Definisce un blocco di dati inizializzati che devono essere caricati in memoria e possibilmente rilocati. Il linker non deve alterare questi blocchi se fanno parte di un nodo di overlay, visto che possono essere letti nuovamente da disco durante la gestione dell'overlay. La tavola che segue ne riporta il formato.

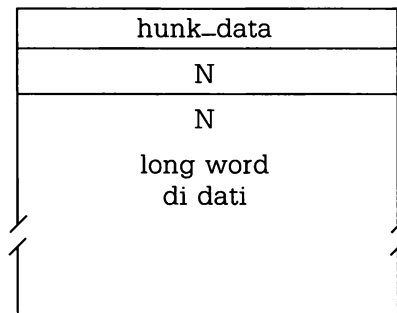



Tavola 2-D: hunk_data (1002/3EA)

2.2.5 hunk_bss (1003/3EB)

Individua un blocco costituente un'area di lavoro non inizializzata che viene allocata dal loader. I blocchi BSS sono usati per stack o simili e per i

blocchi COMMON del FORTRAN. Non è possibile operare rilocazioni all'interno di un blocco BSS, ma possono essere definiti i simboli. Il formato del blocco appare nella seguente tavola.

hunk_bss
N

Tavola 2-E: hunk_bss (1003/3EB)

N è la dimensione, in long word, del blocco richiesto. La memoria usata dai blocchi BSS viene azzerata dal loader quando è allocata.

Il blocco rilocabile all'interno di un hunk deve essere hunk_code, hunk_data oppure hunk_bss.

2.2.6 hunk_reloc32 (1004/3EC)

Un blocco hunk_reloc32 specifica la rilocazione a 32 bit che deve essere svolta dal linker all'interno del blocco rilocabile corrente. L'informazione di rilocazione si riferisce a una locazione all'interno dell'hunk corrente o di qualunque altro compreso nell'unità di programma. Ogni hunk di un'unità di programma è numerato partendo da zero. Il linker aggiunge l'indirizzo di base dell'hunk specificato a ognuna delle long word nel precedente blocco rilocabile che la lista di offset ha indicato. La lista di offset, la cui fine viene indicata da uno zero, include soltanto gli hunk che hanno un riferimento. Il suo formato è riportato nella tavola della pagina successiva.

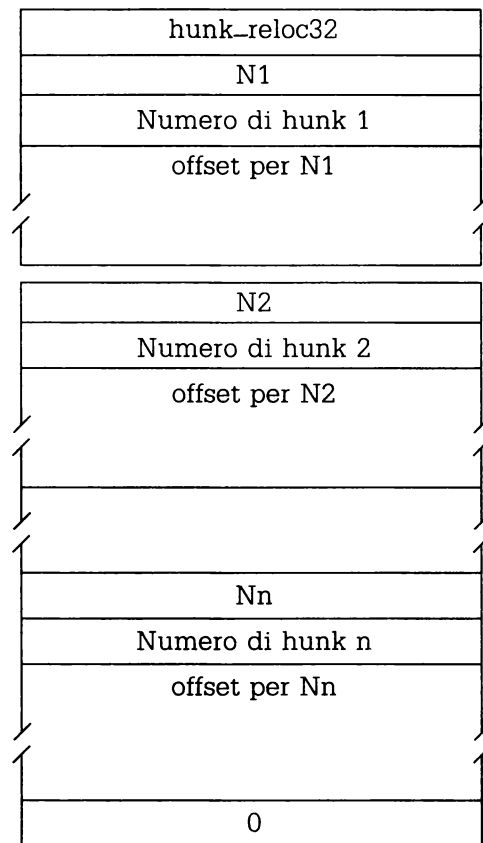



Tavola 2-F: hunk_reloc32

2.2.7 hunk_reloc16 (1005/3ED)

Un blocco hunk_reloc16 specifica la rilocazione a 16 bit che il linker deve eseguire all'interno del blocco rilocabile corrente. L'informazione di rilocazione indica un riferimento a 16 bit rispetto al PC ad altri hunk nell'unità di programma. Il formato è lo stesso dei blocchi hunk_reloc32. I riferimenti devono essere all'hunk dello stesso nome in modo che il linker possa effettuare la rilocazione, mentre raggruppa gli hunk con nomi uguali.

2.2.8 hunk_reloc8 (1006/3EE)

Un blocco hunk_reloc8 specifica la rilocazione a 8 bit che il linker deve eseguire all'interno del blocco rilocabile corrente. L'informazione di rilocazione indica un riferimento a 8 bit relativo al PC ad altri hunk nell'unità di programma. Il formato è lo stesso dei blocchi hunk_reloc32. I riferimenti devono essere verso hunk con lo stesso nome, in modo che il linker possa effettuare la rilocazione mentre raggruppa gli hunk con nomi uguali.

2.2.9 hunk_ext (1007/3EF)

Questo blocco contiene informazioni sui simboli esterni. Contiene voci che definiscono sia simboli sia liste di riferimenti a essi. Il formato è quello rappresentato nella tavola che segue.

hunk_ext
Unità dati del simbolo
Unità dati del simbolo
0

Tavola 2-G: hunk_ext (1007/3EF)

Nel formato è presente una "unità dati del simbolo" per ogni simbolo usato e il blocco termina con una word a zero.

Ogni unità dati del simbolo consiste in un byte indicante il tipo, la lunghezza del nome del simbolo (tre byte), il nome stesso del simbolo e ulteriori dati. La lunghezza del nome del simbolo è data in long word e il campo del nome deve essere allineato, con eventali byte a zero, a una long word.

Il byte del tipo indica se si tratta di una definizione o di un riferimento a un simbolo, e così via. L'AmigaDOS usa i valori tra 0 e 127 per le definizioni dei simboli e quelli tra 128 e 255 per i riferimenti.

I valori sono quelli riportati nella tavola della pagina successiva.

Nome	Valore	Significato
ext_symb	0	Tavola dei simboli - vedere il blocco dei simboli più avanti
ext_def	1	Definizione rilocabile
ext_abs	2	Definizione assoluta
ext_res	3	Definizione libreria residente
ext_ref32	129	Riferimento da 32 bit a un simbolo
ext_common	130	Riferimento da 32 bit a COMMON
ext_ref16	131	Riferimento da 16 bit a un simbolo
ext_ref8	132	Riferimento da 8 bit a un simbolo

Tavola 2-H: simboli esterni

Il linker rifiuta tutti gli altri valori. Per ext_def c'è una word di dati che è il valore del simbolo. Questo è soltanto l'offset del simbolo dall'inizio dell'hunk. Per ext_abs c'è solo un dato, cioè il valore assoluto che deve essere aggiunto nel codice. Il linker tratta il valore di ext_res nella stessa maniera di ext_def, a parte il fatto che il nome dell'hunk è assunto come nome della libreria e che viene copiato nel file caricabile. I tipi byte ext_ref32, ext_ref16 ed ext_ref8 sono seguiti da un valore e da una lista di riferimenti indicati come offset rispetto all'inizio dell'hunk.

Il tipo ext_common ha la stessa struttura, a parte il fatto che ha un blocco di dimensione COMMON prima del valore. Il linker tratta i simboli definiti come COMMON nel modo seguente: se incontra una definizione di un simbolo riferito come COMMON, usa questo valore (l'unica volta che una definizione di questo tipo può apparire è in un blocco di dati FORTRAN), altrimenti alloca un appropriato spazio BSS usando la massima dimensione che è stata specificata per un riferimento a un simbolo COMMON.

Il linker gestisce i riferimenti esterni in modo differente, a seconda del tipo di definizione corrispondente. Somma i valori assoluti alle long word o ai campi di byte e genera un errore se il valore con segno non va bene. I riferimenti rilocabili a 32 bit hanno il valore del simbolo sommato al campo, e per il loader viene prodotto un record di rilocazione. I riferimenti a 16 e 8 bit sono gestiti come riferimenti relativi al PC e possono essere effettuati soltanto verso hunk dotati dello stesso nome, in maniera tale che i diversi hunk siano opportunamente organizzati dal linker prima che vengano caricati. I riferimenti relativi al PC possono fallire se la definizione e il riferimento sono troppo distanti. Il linker può accedere alle definizioni delle librerie residenti soltanto con riferimenti a 32 bit, che vengono gestiti come riferimenti rilocabili a 32 bit. I formati delle unità dati dei simboli sono rappresentati nella tavola riportata nella pagina successiva.

ext_def/abs/res

ext_ref32/16/8

ext_common

Tavola 2-I: unità dati del simbolo

2.2.10 hunk_symbol (1008/3F0)

Questo blocco si usa per far seguire un hunk da una tavola di simboli in modo che si possa utilizzare per il codice un debugger simbolico. Il linker passa attraverso i blocchi delle tavole dei simboli collegati all'hunk e, se gli hunk sono raggruppati, provvede a mettere insieme anche le varie tavole dei simboli. Il loader non trasferisce in memoria questo tipo di blocchi; è il debugger che ne fa uso. Il formato del blocco di una tavola di simboli è identico a quello adottato per il blocco contenente le informazioni dei simboli esterni, con unità contenenti la tavola dei simboli per ogni nome usato. Come tipo del codice si usa zero all'interno delle unità dati dei simboli. Il valore del simbolo è, come al solito, l'offset di tale simbolo rispetto all'inizio dell'hunk. Il formato è quindi quello che appare nella seguente tavola.

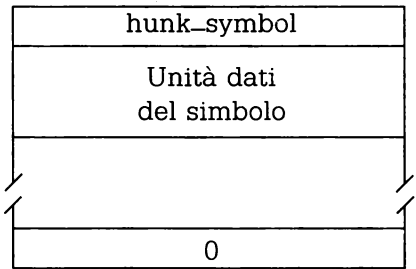



Tavola 2-J: hunk_symbol (1008/3F0)

Nella tavola precedente ogni unità dati del simbolo ha il seguente formato:

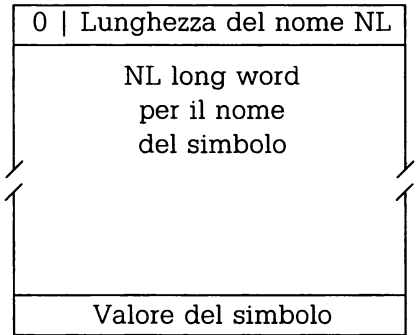



Tavola 2-K: unità dati del simbolo

2.2.11 hunk_debug (1009/3F1)

L'AmigaDOS fornisce un blocco di debug in modo che un file oggetto possa recare con sé ulteriori informazioni sulla procedura di debugging. Per esempio un compilatore di un linguaggio ad alto livello potrebbe aver bisogno di mantenere le descrizioni delle strutture dati per utilizzare debugger ad alto livello. Il blocco di debug mantiene questo tipo di informazioni. L'AmigaDOS non impone un formato per il blocco di debug se si eccettua il fatto che deve iniziare con la long word hunk_debug seguita da una long word che contiene la dimensione, in long word, del blocco. Il formato è riportato nella tavola che segue.

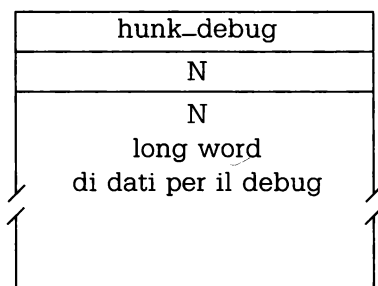



Tavola 2-L: hunk_debug (1009/3F1)

2.2.12 hunk_end (1010/3F2)

Specifica la fine di un hunk ed è composto da una sola long word: hunk_end.

2.3 File caricabili

Il formato di un file caricabile (cioè l'output generato dal linker) è simile a quello di un file oggetto. In particolare identifica un certo numero di hunk con un formato simile a quello di un file oggetto. La differenza principale risiede nel fatto che gli hunk non contengono mai un blocco di informazioni sui simboli esterni, visto che tutti i simboli esterni sono stati risolti, e che non sono incluse le informazioni circa l'unità di programma. In un semplice file caricabile, cioè senza overlay, c'è un blocco di intestazione che indica il numero totale di hunk presenti e qualunque libreria residente cui il programma si riferisce. Questo blocco è seguito dagli hunk, che possono essere il risultato del raggruppamento di una serie di hunk in input che possiedono lo stesso nome. Questa intera struttura è conosciuta come un

nodo. I file caricabili possono anche contenere informazioni di overlay; in questo caso esiste una tavola di overlay collocata subito dopo il primo nodo, seguita dai nomi degli overlay separati da uno speciale blocco di break. Quindi la struttura di un file caricabile può essere riassunta come segue (le voci contrassegnate con un asterisco sono opzionali).

- Nodo primario
- Blocco della tabella di overlay (*)
- Nodi di overlay separati dai blocchi di break (*)

I blocchi di rilocazione all'interno degli hunk sono sempre del tipo hunk_reloc32 e indicano la rilocazione che deve essere eseguita al momento del caricamento. Questa operazione riguarda sia la rilocazione a 32 bit specificata con i blocchi hunk_reloc32 nel file oggetto, sia l'ulteriore rilocazione richiesta per la risoluzione dei simboli esterni.

Ogni riferimento esterno nel file oggetto viene gestito nel modo seguente: il linker ricerca nell'input primario una definizione esterna corrispondente. Se non ne trova, estende la ricerca nelle librerie associate e include l'intera unità programma dove è avvenuta la definizione. Nel corso di questa operazione possono generarsi ulteriori definizioni insolute. Al termine della prima fase, il linker conosce tutte le definizioni esterne e il numero totale di hunk che saranno usati. Questi hunk comprendono quelli all'interno del file caricabile e quelli associati alle librerie residenti. Il linker, durante la seconda fase, predispone i riferimenti esterni a 32 bit in modo che indichino l'offset richiesto all'interno dell'hunk che definisce il simbolo. Il linker produce una voce extra nel blocco di rilocazione in modo da poter aggiungere a ogni riferimento esterno, quando gli hunk vengono caricati, l'indirizzo base dell'hunk che definisce il simbolo. Questo meccanismo è adottato anche per le librerie residenti.

Prima che il loader possa operare questi riferimenti incrociati tra hunk, ha bisogno di conoscere il numero e la dimensione dei vari hunk nei nodi. Il blocco di intestazione fornisce queste informazioni, come verrà descritto successivamente. Il file caricabile può anche contenere le informazioni di overlay in un blocco appropriato; i blocchi di break separano i nodi di overlay.

2.3.1 hunk_header (1011/3F3)

Fornisce le informazioni riguardanti il numero di hunk che devono essere caricati e la dimensione di ognuno; contiene inoltre il nome di qualunque libreria residente che deve essere aperta prima che il nodo sia caricato.

Il formato di hunk_header è descritto nella Tavola 2-M. La prima parte del blocco di intestazione contiene i nomi delle librerie residenti che il loader deve aprire quando questo nodo viene caricato. Ogni nome consiste in una long word che indica la lunghezza del nome espressa in long word, e nel testo

del nome, allineato alla long word per mezzo di caratteri nulli. I nomi sono già nell'ordine che il loader deve seguire per aprire le librerie corrispondenti.

Quando il loader carica un nodo primario, alloca in memoria una tavola che usa per rintracciare tutti gli hunk che ha caricato. Questa tavola deve essere abbastanza grande per tutti gli hunk del file caricabile, compresi gli hunk in overlay. Il loader impiega questa tavola anche per mantenere una copia delle tavole di hunk associate a ogni libreria residente. La long word successiva nel blocco di intestazione indica proprio la dimensione della tavola, che è uguale al numero dell'hunk massimo più uno.

La long word F si riferisce al primo slot della tavola degli hunk che il loader deve usare quando carica. Per un nodo primario che non si riferisce a una libreria residente questo valore è zero; altrimenti è pari al numero di hunk delle librerie residenti. Il loader copia le voci dalla tavola degli hunk associata a una libreria in seguito a una richiesta di apertura di libreria. Per un nodo overlay questo valore è pari al numero degli hunk delle librerie residenti, più il numero degli hunk già caricati dai nodi, che lo precedono gerarchicamente.

La long word L si riferisce all'ultimo slot della tavola che il loader deve usare: il numero totale degli hunk è quindi pari a L-F+1.

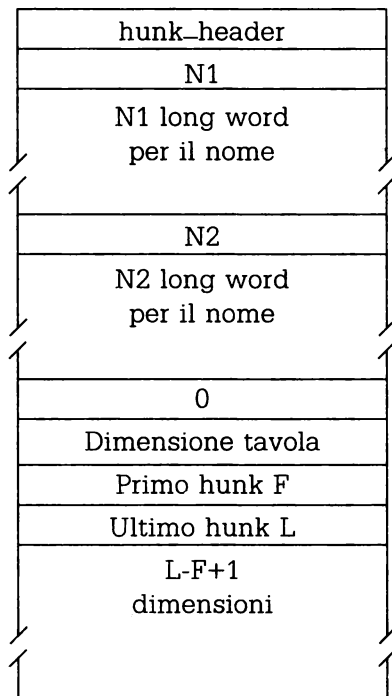



Tavola 2-M: hunk_header (1011/3F3)

Il blocco di intestazione continua con L-F+1 long word che indicano la dimensione di ogni hunk che deve essere caricato. Queste informazioni permettono al loader di pre-allocare lo spazio per gli hunk e quindi di eseguire la rilocazione richiesta tra gli hunk quando vengono caricati. Un hunk potebbe essere del tipo BSS con una dimensione di zero; in questo caso il loader usa una variabile del sistema operativo per fornire la dimensione come descritto in hunk_bss a pagina 303.

2.3.2 hunk_overlay (1013/3F5)

Il blocco della tavola di overlay contiene tutti i dati necessari per indicare al loader che sta caricando un programma in overlay. Quando il loader incontra questo blocco, appronta la tavola e ritorna, lasciando il canale di input relativo al file ancora aperto. Il suo formato è quello riportato nella seguente tavola.

hunk_overlay
Dimensione tavola
M+2
M+1 zeri
Tavola dei dati di overlay

Tavola 2-N: hunk_overlay (1013/3F5)

La prima long word è il limite superiore (in long word) della completa tavola di overlay.

M è il massimo livello dell'albero di overlay usato, e il livello della radice viene considerato livello zero. Le successive M+1 word formano la tavola delle ordinate dello schema di overlay.

Il resto del blocco è occupato dai dati della tavola di overlay: una serie di voci composte da otto word, una per ogni simbolo overlay. Se il massimo numero d'overlay usato è zero, la dimensione della tavola è (0+1)*8. Quindi la dimensione della tavola di overlay è uguale a (0+1)*8+M+1.

2.3.3 hunk_break (1014/3F6)

Un blocco di break indica la fine di un nodo di overlay ed è composto da una sola long word: hunk_break.

2.4 Esempi

La semplice sezione di codice seguente illustra come il linker e il loader gestiscono i simboli esterni. Per esempio,

```
                                        IDNT      A
                                        XREF      BILLY,JOHN
                                        XDEF      MARY
* La long word successiva richiede la rilocazione
0000' 0000 0008                         DC.L      FRED
0004' 123C 00FF                         MOVE.B    #$FF,D1
0008' 7001                        FRED  MOVEQ     #1,D0
* Punto d'entrata esterno
000A' 4E71                        MARY  NOP
000C' 4EB9 0000 0000                    JSR       BILLY          Chiamata esterna
0012' 2239 0000 0000                    MOVE.L    JOHN,D1        Riferimento esterno
                                        END
```

produce il seguente file oggetto:

```
hunk_unit
00000001   Dimensione, in long word, del nome
41000000   Nome (''A''), riempito di zeri per l'allineamento
hunk_code
00000006   Dimensione, in long word, del codice
00000008 123C00FF 70014E71 4EB90000 00002239 00000000
hunk_reloc32
00000001   Numero nell'hunk 0
00000000   hunk 0
00000000   Offset da rilocare
00000000   Zero per segnalare la fine
hunk_ext
01000001   XDEF, una long word di lunghezza
4D415259   MARY
0000000A   Offset della definizione
81000001   XREF, una long word di lunghezza
4A4F484E   JOHN
00000001   Numero di riferimenti
```

```
00000014   Offset del riferimento
81000002   XREF, due long word di lunghezza
42494C4C   BILLY
59000000   (zeri per riempire)
00000001   Numero di riferimenti
0000000E   Offset del riferimento
00000000   Fine del blocco esterno
hunk_end
```

Il programma associato al precedente è il seguente:

```
                                     IDNT      B
                                     XDEF      BILLY,JOHN
                                     XREF      MARY
0000' 2A3C AAAA AAAA                 MOVE.L    #$AAAAAAAA,D5
* Punto d'entrata esterno
0006' 4E71                     BILLY NOP
* Punto d'entrata esterno
0008' 7201                     JOHN  MOVEQ     #1,D1
* Chiamata a riferimento esterno
000A' 4EF9 0000 0000                 JMP       MARY
                                     END
```

e il corrispondente codice oggetto è:

```
hunk_unit
00000001   Dimensione, in long word, del nome
42000000   Nome dell'unita'
hunk_code
00000004   Dimensione, in long word, del codice
2A3CAAAA AAAA4E71 72014EF9 00000000
hunk_ext
01000001   XDEF, una long word di lunghezza
4A4F484E   JOHN
00000008   Offset della definizione
01000002   XDEF, due long word di lunghezza
42494C4C   BILLY
59000000   (zeri per riempire)
00000006   Offset della definizione
81000001   XREF, una long word di lunghezza
4D415259   MARY
00000001   Numero di riferimenti
0000000C   Offset del riferimento
00000000   Fine del blocco di esterni
hunk_end
```

Una volta che viene fatto passare attraverso il linker, il file oggetto assume il seguente formato:

```
hunk_header
00000000   Nessun nome di hunk
00000002   Dimensione della tavola degli hunk
00000000   Primo hunk
00000001   Ultimo hunk
00000006   Dimensione dell'hunk 0
00000004   Dimensione dell'hunk 1
hunk_code
00000006   Dimensione del codice, in long word
00000008 123C00FF 70014E71 4EB90000 00062239 00000008
hunk_reloc32
00000001   Numero nell'hunk 0
00000000   hunk 0
00000000   Offset da rilocare
00000002   Numero nell'hunk 1
00000001   hunk 1
00000014   Offset da rilocare
0000000E   Offset da rilocare
00000000   Zero per indicare la fine
hunk_end
hunk_code
00000004   Dimensione del codice, in long word
2A3CAAAA AAAA4E71 72014EF9 0000000A
hunk_reloc32
00000001   Numero nell'hunk 0
00000000   hunk 0
0000000C   Offset da rilocare
00000000   Zero per indicare la fine
hunk_end
```

Quando il loader carica in memoria questo codice, legge il blocco di intestazione e alloca una tavola degli hunk di due long word. In seguito alloca lo spazio, chiamando una routine del sistema operativo, per due aree lunghe rispettivamente 6 e 4 long word. Supponendo che le due aree si trovino alla locazione 3000 e alla 7000, la tavola degli hunk conterrà i valori 3000 e 7000.

Il loader legge il primo hunk e inserisce il codice a partire da 3000; poi esegue la rilocazione. La prima voce specifica la rilocazione rispetto all'hunk 0, così viene sommato 3000 alla long word con offset 0 convertendo il valore memorizzato da 00000008 a 00003008. La seconda voce indica la rilocazione rispetto all'hunk 1. Anche se non è stato ancora caricato, si sa che verrà

memorizzato a partire dalla locazione 7000; quindi 7000 sarà sommato al valore memorizzato in 300E e 3014. Occorre notare che il linker ha già inserito gli offset 00000006 e 00000008 nei riferimenti dell'hunk 0, in modo che indichino per la definizione gli offset corretti nell'hunk 1. Così le long word che indicano i riferimenti esterni finiscono per contenere i valori 00007006 e 00007008, che sono le locazioni corrette quando il secondo hunk sarà caricato.

Il secondo hunk viene caricato in memoria nella stessa maniera del primo a partire dalla locazione 7000. Le informazioni di rilocazione fanno in modo che la long word posta a 700C passi da 0000000A (l'offset di MARY nel primo hunk) a 0000300A (l'indirizzo di MARY in memoria).

Il loader gestisce i riferimenti alle librerie residenti nello stesso modo, con l'eccezione che, dopo aver aperto la libreria, copia le locazioni degli hunk contenuti nella libreria all'inizio della tavola degli hunk. In seguito imposta i riferimenti alla libreria residente in modo che indichino la locazione corretta, aggiungendo la base degli hunk della libreria.

Capitolo 3

Strutture dati dell'AmigaDOS

Questo capitolo descrive le strutture dati dell'AmigaDOS in memoria e nei file. Non descrive il formato di un disco, che è illustrato nel capitolo 1.

3.1 Introduzione
3.2 Struttura dati del processo
3.3 Struttura dati Global
3.3.1 Sottostruttura Info
3.4 Allocazione della memoria
3.5 Liste di segmenti
3.6 Handle dei file
3.7 Lock
3.8 Packet
3.8.1 Tipi di packet

3.1 Introduzione

L'AmigaDOS gestisce l'input e l'output indipendentemente dalla periferica correntemente selezionata, creando un opportuno processo di gestione per ogni device che viene utilizzato. Il processo di gestione accetta un insieme standard di richieste di I/O e le converte, quando si rende necessario, in richieste specifiche per quel tipo di device. Tutti i task dell'AmigaDOS si riferiscono al processo di gestione piuttosto che direttamente al device, anche se l'accesso diretto è realizzabile in caso di necessità. Questo capitolo descrive le strutture dati dell'AmigaDOS, compresi il formato di un processo, le strutture dati centrali condivise e la struttura delle richieste da inviare al gestore dei device.

In aggiunta ai normali valori dell'Amiga, come LONG e APTR, l'AmigaDOS usa i BPTR. Un BPTR è un puntatore BCPL, cioè un puntatore a un blocco di memoria diviso per quattro allineato a una long word. Così, per leggere un BPTR in C, si opera semplicemente uno scorrimento a sinistra di 2. Per creare un BPTR si possono usare sia la memoria ottenuta da una chiamata

ad AllocMem, sia una struttura sul proprio stack, quando si è certi che sullo stack sono state allocate solo long word (lo stack iniziale è allineato secondo long word successive). Si deve poi operare uno shift a destra di 2 per creare il BPTR.

L'AmigaDOS usa anche un BSTR, che è una stringa BCPL. Un BSTR consiste in un BPTR che individua le locazioni di memoria contenenti la lunghezza della stringa nel primo byte e la stringa nei byte successivi.

In questo capitolo appaiono un certo numero di riferimenti al Global Vector (vettore globale). Il Global Vector è una tavola di salti usata dal BCPL, ed è un vettore globale condiviso standard individuato da GlobVec. Alcuni processi, come il gestore dei file, usano un vettore globale privato.

I file include dos.h e dosextens.h contengono le definizioni, in linguaggio C, delle seguenti strutture, mentre i corrispondenti file .i sono per il linguaggio Assembly.

3.2 Struttura dati del processo

Questi valori sono creati come parte di un processo dell'AmigaDOS, e ogni processo ne possiede un insieme completo.

Un processo è un task dell'Exec con un certo numero di strutture dati aggiuntive. La struttura di un processo consiste in:

- struttura task dell'Exec
- message port dell'Exec
- valori di processo dell'AmigaDOS

L'identificatore di processo che l'AmigaDOS usa internamente è un puntatore alla message port dell'Exec (dalla quale si ottiene il task Exec).

I valori di processo dell'AmigaDOS sono i seguenti:

Valore	**Funzione**	**Descrizione**
BPTR	SegArray	Array di SegList usato da questo processo
LONG	StackSize	Dimensione, in byte, dello stack assegnato al processo
APTR	GlobVec	Global Vector per questo processo
LONG	TaskNum	Numero di task CLI, oppure zero se non è CLI
BPTR	StackBase	Puntatore alla fine alta della memoria dello stack assegnato al processo
LONG	IoErr	Valore del risultato secondario dell'ultima chiamata
BPTR	CurrentDir	Lock associato alla directory corrente
BPTR	CIS	Canale di input corrente del CLI
BPTR	COS	Canale di output corrente del CLI

Valore	Funzione	Descrizione
APTR	CoHand	Processo che gestisce la console della finestra corrente
APTR	FiHand	Processo che gestisce i file del device corrente
BPTR	CLIStruct	Puntatore a informazioni CLI aggiuntive
APTR	ReturnAddr	Puntatore allo stackframe precedente
APTR	PktWait	Funzione da chiamare quando si attendono dei messaggi
APTR	WindowPtr	Puntatore alla finestra

Per identificare il segmento di memoria che un particolare processo sta usando, si utilizza SegArray. SegArray è un array di long word la cui dimensione è contenuta in SegArray[0]. Gli altri elementi sono zero oppure BPTR a una SegList. CreateProc crea questo array con i primi due elementi dell'array che puntano al codice residente e il terzo elemento che è la SegList passata come argomento. Quando un processo termina, viene usato FreeMem per liberare lo spazio di SegArray.

StackSize indica la dimensione dello stack del processo, come specificato dal programmatore quando chiama CreateProc. Bisogna notare che lo stack del processo non è lo stack dei comandi che il CLI usa quando richiama un programma. Il CLI ottiene il proprio stack dei comandi prima di eseguire un programma e la sua dimensione può essere variata con il comando STACK. L'AmigaDOS, quando crea un processo, ne ottiene lo stack e ne memorizza la dimensione in StackSize. Anche il puntatore allo spazio per il blocco di controllo del processo e allo stack è memorizzato nel campo MemEntry della struttura task. Quando il processo termina, questo spazio viene liberato per mezzo di una chiamata a FreeMem. È anche possibile includere nella struttura altre aree di memoria in maniera che vengano disallocate quando il task termina.

Se una chiamata a CreateProc crea il processo, il puntatore GlobVec individua il Global Vector condiviso. Alcuni processi interni di gestione usano un GlobVec privato.

Il valore di TaskNum è normalmente zero; un processo CLI memorizza qui il numero intero che identifica la chiamata al CLI.

StackBase è un puntatore alla fine alta dello stack assegnato al processo. Questa è la fine dello stack per linguaggi come C e Assembly, mentre è l'inizio per linguaggi come il BCPL.

I valori di IoErr e CurrentDir sono quelli gestiti dalle funzioni dell'Amiga-DOS con nome a loro simile. CIS e COS sono i valori restituiti da Input e Output e si riferiscono agli handle dei file che si devono usare quando si esegue un programma in ambiente CLI. Negli altri casi CIS e COS sono zero.

CoHand e FiHand si riferiscono al gestore della console della finestra corrente e al gestore dei file del device corrente. Questi valori si usano

quando si cerca di aprire il device * o un file attraverso un relativo nome con path.

Il puntatore CLIStruct è diverso da zero solo per i processi CLI. In questo caso si riferisce a un'ulteriore struttura usata dal CLI, che ha il seguente formato:

Valore	Funzione	Descrizione
LONG	Result2	Valore di IoErr dell'ultimo comando
BSTR	SetName	Nome della directory corrente
BPTR	CommandDir	Lock associato alla directory dei comandi
LONG	ReturnCode	Codice di ritorno dell'ultimo comando
BSTR	CommandName	Nome del comando corrente
LONG	FailLevel	Livello di fallimento (impostato da FAILAT)
BSTR	Prompt	Prompt corrente (indicato da PROMPT)
BPTR	StandardIn	Input CLI di default (il terminale)
BPTR	CurrentIn	Input CLI corrente
BSTR	CommandFile	Nome del file di comandi EXECUTE
LONG	Interactive	Booleano; vero se si richiede il prompt
LONG	Background	Booleano; vero se il CLI è stato creato con il comando RUN
BPTR	CurrentOut	Output CLI corrente
LONG	DefaultStack	Dimensione dello stack (in long word) che si vuole ottenere
BPTR	StandardOut	Output CLI di default (il terminale)
BPTR	Module	SegList del comando attualmente in memoria

La funzione Exit usa il valore di ReturnAddr che punta appena prima dell'indirizzo di ritorno memorizzato nello stack in quel momento attivo. Se il programma termina eseguendo un RTS su uno stack vuoto, il controllo passa al codice il cui indirizzo è stato inserito nello stack da CreateProc o dal CLI. Se un programma termina chiamando Exit, l'AmigaDOS utilizza questo puntatore per ottenere lo stesso indirizzo di ritorno.

Il valore di PktWait è di solito zero. Se è diverso da zero, l'AmigaDOS chiama PktWait ogni volta che un processo deve essere sospeso per attendere la segnalazione dell'arrivo di un messaggio. La funzione dovrebbe restituire un messaggio nello stesso modo di GetMsg, appena ne è disponibile uno. Di solito si usa questa funzione per filtrare qualunque messaggio privato – non inteso per l'AmigaDOS – che arrivi alla message port standard del processo.

Si usa il valore di WindowPtr quando l'AmigaDOS riscontra un errore che normalmente richiede un'azione da parte dell'utente. Esempi di questi errori sono il tentativo di scrivere su un disco protetto in scrittura oppure pieno. Se il valore di WindowPtr è -1, l'errore viene segnalato al programma sotto forma di codice d'errore generato da una chiamata alle funzioni AmigaDOS Open,

Write o altro. Se questo valore è zero, l'AmigaDOS visualizza un requester sullo schermo del Workbench informando l'utente dell'errore e fornendo la possibilità di ripetere l'operazione oppure di cancellarla. Se l'utente seleziona "cancel", l'AmigaDOS restituisce il codice d'errore al programma che chiama; se invece si seleziona "retry", oppure si inserisce un disco, l'AmigaDOS tenta di compiere un'altra volta l'operazione.

Memorizzando un valore positivo nel campo WindowPtr, l'AmigaDOS lo considera un puntatore a una struttura Window. Normalmente si usa l'indirizzo della struttura Window della finestra corrente, in modo che l'AmigaDOS visualizzi i messaggi all'interno della finestra piuttosto che sullo schermo del Workbench. Si può lasciare sempre il campo WindowPtr a zero, ma i messaggi che l'AmigaDOS visualizza appaiono sullo schermo del Workbench e di conseguenza possono, nel caso si stia usando un altro schermo, coprire la propria zona di lavoro.

Il valore iniziale di WindowPtr è ereditato dal processo che ha creato quello corrente. Se si decide di alterare il valore di WindowPtr dall'interno di un programma che è eseguito sotto CLI, occorre salvarne il contenuto originale per poterlo ripristinare quando il programma termina; altrimenti il processo CLI contiene un WindowPtr che si riferisce a una finestra non più presente.

3.3 Struttura dati Global

Questa struttura dati è presente una volta sola, tuttavia viene usata da tutti i processi AmigaDOS. Se si invoca la funzione OpenLibrary, si ottiene un puntatore alla base della libreria. La base della struttura dati è un offset positivo rispetto al puntatore alla base della libreria. Il puntatore alla base della libreria individua la seguente struttura:

- struttura Library Node
- APTR al RootNode DOS
- APTR al Shared Global Vector DOS (vettore globale condiviso)
- copia dei registri privati del DOS

Tutte le funzioni interne dell'AmigaDOS usano il vettore globale condiviso, che è una tavola di salti. Normalmente non si dovrebbe usarlo, se non tramite le chiamate dirette all'apposita interfaccia, dal momento che può essere cambiato senza avvertimento.

La struttura RootNode è la seguente:

Valore	Funzione	Descrizione
BPTR	TaskTable	Array dei processi attualmente in esecuzione
BPTR	CLISegList	SegList per il CLI
LONG	Days	Numero dei giorni nel tempo attuale

Valore	Funzione	Descrizione
LONG	Mins	Numero dei minuti nel tempo attuale
LONG	Ticks	Numero dei tick nel tempo attuale
BPTR	RestartSeg	SegList per il processo di convalida dei dischi
BPTR	Info	Puntatore alla sottostruttura Info

TaskTable è un array la cui dimensione è mantenuta in TaskTable[0]. L'id di processo (in altre parole la MsgPort associata al processo) di ogni CLI è memorizzato nell'array. L'id del CLI con TaskNum "n" è mantenuto in TaskTable[n]; una posizione vuota è riempita con uno zero. I comandi RUN e NEWCLI scandiscono TaskTable per identificare una posizione libera, da usare come TaskNum del CLI appena creato.

CLISegList è il SegList per il codice del CLI. RUN e NEWCLI usano questo valore per creare una nuova ricorrenza del CLI.

RootNode contiene l'ora e la data corrente; normalmente si dovrebbe usare la funzione dell'AmigaDOS DateStamp per ottenere un insieme valido di valori. I valori Days, Mins e Ticks indicano appunto la data e l'ora. Il valore di Days indica il numero di giorni trascorsi dall'1 gennaio 1978. Il valore di Mins è il numero di minuti passati da mezzanotte. Un tick equivale a un cinquantesimo di secondo, ma questo valore è aggiornato una sola volta al secondo.

RestartSeg è il SegList per il codice del processo di convalida dei dischi, un processo che viene creato dall'AmigaDOS ogni volta che si inserisce un nuovo disco in un drive.

3.3.1 Sottostruttura Info

Per accedere a un'ulteriore sottostruttura con il seguente formato si usa il puntatore Info.

Valore	Funzione	Descrizione
BPTR	McName	Nome di Network di questa macchina; attualmente zero
BPTR	DevInfo	Lista dei device
BPTR	Devices	Attualmente zero
BPTR	Handlers	Attualmente zero
APTR	NetHand	L'id di processo del gestore del network; attualmente zero

La maggior parte dei campi della sottostruttura Info sono, al momento, vuoti, ma la Commodore-Amiga ha intenzione di usarli per espandere il sistema.

La struttura DevInfo è una lista concatenata che viene usata per

identificare tutti i nomi di device riconosciuti dall'AmigaDOS. Tra questi rientrano i nomi assegnati con il comando ASSIGN e quelli dei volumi dei dischi. Vi sono due formati possibili per le voci della lista, rispettivamente adottabili qualora la voce si riferisca a un volume di disco oppure no. Il formato di una voce che descrive un device oppure una directory (tramite ASSIGN) è il seguente:

Valore	**Funzione**	**Descrizione**
BPTR	Next	Puntatore alla voce successiva oppure zero
LONG	Type	Tipo della voce (device o dir)
APTR	Task	Processo di gestione oppure zero
BPTR	Lock	Lock del file system oppure zero
BSTR	Handler	Nome del file di gestione oppure zero
LONG	StackSize	Dimensione dello stack per il processo di gestione
LONG	Priority	Priorità del processo di gestione
LONG	Startup	Valore di startup da passare al processo di gestione
BPTR	SegList	SegList per il processo di gestione oppure zero
BPTR	GlobVec	Global Vector del processo di gestione oppure zero
BSTR	Name	Nome del device o nome assegnato tramite ASSIGN

Il campo Next collega insieme tutte le voci della lista, mentre il nome del device logico è mantenuto nel campo Name.

Il campo Type è 0 (dt_device) oppure 1 (dt_dir). Si può creare una voce di directory con il comando ASSIGN, che assegna a una directory un nome, il quale viene poi usato come nome di device. Se la voce indica una directory, allora Task si riferisce al processo di gestione del file system relativo a quel disco e il campo Lock contiene un puntatore al lock di quella directory.

Nel caso che la voce si riferisca a un device occorre distinguere tra quelli che sono residenti e quelli che non lo sono. Se il device è residente, Task identifica il processo di gestione e Lock è normalmente a zero. Se il device non è residente, allora Task è a zero e l'AmigaDOS utilizza il resto della struttura.

Se SegList è a zero significa che il codice del device non è in memoria. Il campo Handler è una stringa la quale individua il file che contiene il codice (per esempio SYS:L/RAM-HANDLER). Una chiamata alla funzione LoadSeg carica il codice in memoria e inserisce il risultato nel campo SegList.

L'AmigaDOS crea quindi un nuovo processo di gestione con i valori SegList, StackSize e Pri. Il nuovo processo è un processo BCPL e richiede un Global Vector, che può essere il valore specificato in GlobVec.oppure un

nuovo vettore globale privato se il campo è zero.

Al nuovo processo viene mandato un messaggio che contiene il nome indicato originariamente, il valore memorizzato in Startup e la base della voce della lista. Il nuovo processo di gestione può poi decidere se inserire o meno l'id di processo in qualche locazione della TaskTable. Se questa locazione viene aggiornata, i riferimenti seguenti al nome del device usano lo stesso task di gestione: questo è quanto viene svolto dal device RAM:. Se la locazione non viene riempita, gli ulteriori riferimenti al device causano la ripetizione del processo: questo è quanto viene svolto dal device CON:.

Il formato della struttura è leggermente differente se il campo Type è uguale a 2 (dt_volume).

Valore	**Funzione**	**Descrizione**
BPTR	Next	Puntatore alla voce successiva oppure zero
LONG	Type	Tipo della voce (volume)
APTR	Task	Processo di gestione oppure zero
BPTR	Lock	Lock del file system
LONG	VolDays	Data di creazione del volume
LONG	VolMins	
LONG	VolTicks	
BPTR	LockList	Lista dei lock attivi di questo volume
LONG	DiskType	Tipo del disco
LONG	Spare	Non usato
BSTR	Name	Nome del volume

In questo caso, il campo del nome è il nome del volume e il campo Task si riferisce al processo di gestione, se il volume è inserito, oppure contiene uno zero se non lo è. Per distinguere i dischi con lo stesso nome, l'AmigaDOS scrive l'ora e la data di creazione del volume, e in seguito la salva nella struttura a lista. L'AmigaDOS può quindi confrontare i momenti della creazione di volumi differenti ogni volta che è necessario.

Se un volume non è inserito, l'AmigaDOS salva la lista dei lock attivi nel campo LockList. DiskType è usato per identificare il tipo del disco. Attualmente ci sono solo dischi AmigaDOS. Il tipo del disco è lungo fino a quattro caratteri raggruppati in una long word riempita a destra con eventuali caratteri nulli.

3.4 Allocazione della memoria

L'AmigaDOS ottiene tutta la memoria che alloca chiamando la funzione AllocMem fornita dall'Exec. In questo modo riceve strutture come i lock o gli handle dei file, e di solito le ripone nello spazio libero generato con la chiamata a FreeMem. Ogni segmento di memoria allocato dall'AmigaDOS è identificato da un BPTR alla seconda long word della struttura. La prima long

word contiene sempre la lunghezza, in byte, dell'intero segmento. Quindi la struttura della memoria allocata è la seguente:

Valore	Funzione	Descrizione
LONG	BlockSize	Dimensione del blocco di memoria
LONG	FirstData	Primo dato del segmento; il BPTR al blocco punta qui

3.5 Liste di segmenti

Per ottenere una lista di segmenti bisogna chiamare LoadSeg. Il risultato è un BPTR verso la memoria allocata, in modo che la lunghezza del blocco di memoria contenente ogni voce della lista sia posto a -4 rispetto al BPTR. Questa lunghezza è pari alla dimensione di una voce della lista dei segmenti aumentata di 8, permettendo così la presenza del campo di collegamento e di quello della dimensione stessa.

SegList è una lista concatenata attraverso i BPTR che termina con uno zero. Il resto di ogni voce della segment list contiene il codice caricato. Quindi il formato è:

Valore	Funzione	Descrizione
LONG	NextSeg	BPTR al prossimo segmento oppure zero
LONG	FirstCode	Primo valore del file binario

3.6 Handle dei file

Gli handle dei file sono creati dalla funzione dell'AmigaDOS Open e vengono usati come argomenti da altre funzioni tipo Read e Write. L'AmigaDOS li restituisce come puntatori (BPTR) alla seguente struttura:

Valore	Funzione	Descrizione
LONG	Link	Non usato
LONG	Interact	Booleano, TRUE (vero) se è interattivo
LONG	ProcessID	ID del processo di gestione
BPTR	Buffer	Buffer per uso interno
LONG	CharPos	Posizione del carattere per usi interni
LONG	BufEnd	Posizione finale per usi interni
APTR	ReadFunc	Funzione chiamata quando il buffer è esaurito
APTR	WriteFunc	Funzione chiamata quando il buffer è pieno
APTR	CloseFunc	Funzione chiamata quando l'handle è chiuso
LONG	Arg1	Argomento; dipende dal tipo dell'handle
LONG	Arg2	Argomento; dipende dal tipo dell'handle

La maggior parte dei campi vengono usati dall'AmigaDOS solo internamente; di solito Read e Write impiegano l'handle del file per indicare il processo di gestione e qualunque argomento che deve essere passato. I valori all'interno dell'handle non dovrebbero essere modificati dai programmi dell'utente, ad eccezione del primo campo, che può essere usato per collegare gli handle dei file in una singola lista concatenata.

In questa descrizione sono stati adottati per i campi delle strutture dei nomi simbolici che potrebbero non corrispondere con quelli riportati nei file dosextens.h e dosextens.i. Si tratta solo di differenze nella notazione; il tipo dei campi e il loro significato rimangono invariati.

3.7 Lock

Il filing system utilizza ampiamente una struttura dati chiamata lock, che viene impiegata per due scopi diversi. Il primo riguarda il meccanismo di apertura dei file per letture multiple e scritture singole. A questo proposito, è bene notare che ottenere un lock condiviso in lettura di una directory non preclude la possibilità che un altro task vi possa introdurre delle alterazioni.

Il secondo scopo del lock è quello di identificare unicamente un file. Sebbene un particolare file possa essere identificato in varie maniere, il lock è un semplice handle di quel file. Il lock contiene l'indicazione del blocco del disco contenente l'intestazione della directory o del file attuale, ed è di conseguenza uno strumento molto efficiente per specificare un particolare oggetto del file system. La struttura di un lock è la seguente:

Valore	**Funzione**	**Descrizione**
BPTR	NextLock	BPTR al successivo nella catena, altrimenti zero
LONG	DiskBlock	Numero di blocco del File Header o della directory
LONG	AccessType	Accesso esclusivo o condiviso
APTR	ProcessID	ID del processo di gestione
BPTR	VolNode	Dati del volume del lock

Dal momento che l'AmigaDOS usa il campo NextLock per collegare insieme i vari lock, non bisogna alterarlo. Il filing system scrive un valore nel campo DiskBlock, in modo che rappresenti la locazione su disco del blocco della directory o del blocco dell'header del file. AccessType serve per indicare se si tratta di un lock condiviso in lettura, e in questo caso ha un valore pari a -2, o se è un lock esclusivo in scrittura, con valore pari a -1. Il campo ProcessID contiene un puntatore al processo di gestione del device che contiene il file cui il lock si riferisce. Infine il campo VolNode punta al nodo nella struttura DevInfo che identifica il volume al quale si riferisce il

lock. I dati di un volume rimangono nella struttura DevInfo se il disco è inserito e se ci sono lock aperti verso quel volume.

Occorre notare che un lock può anche essere uno zero. Il caso speciale di un lock zero indica che esso si riferisce alla radice iniziale del filing system; quindi il campo FiHand all'interno della struttura dati del processo fornisce il processo di gestione.

3.8 Packet

Il passaggio dei packet gestisce tutte le comunicazioni tra processi effettuate dall'AmigaDOS. Un packet è una struttura costruita sopra il meccanismo di recapito dei messaggi fornito dall'Exec.

Un messaggio (message, in inglese) Exec è una struttura, descritta in *Programmare l'Amiga Vol. I*, dotata di un campo Name. L'AmigaDOS usa questo campo come APTR a un'altra sezione di memoria chiamata packet. Un packet deve essere allineato a una long word e ha la seguente struttura generale.

Valore	**Funzione**	**Descrizione**
APTR	MsgPtr	Puntatore all'indietro verso la struttura message
APTR	MsgPort	Message port dove deve essere inviata la risposta
LONG	PktType	Tipo del packet
LONG	Res1	Primo campo del risultato
LONG	Res2	Secondo campo del risultato
LONG	Arg1	Argomento; dipende dal tipo del packet
LONG	Arg2	Argomento; dipende dal tipo del packet
	...	
LONG	ArgN	Argomento; dipende dal tipo del packet

Il formato specifico di un packet dipende dal tipo di appartenenza, ma in ogni caso contiene: un puntatore all'indietro verso la struttura message, la MsgPort per la risposta e due campi per i risultati. Quando l'AmigaDOS spedisce un packet, la porta di risposta viene sovrascritta dall'identificatore di processo di chi lo ha mandato, in modo che il packet possa essere restituito. Così, ogni volta che si manda un packet a un processo di gestione dell'AmigaDOS, si deve riempire la MsgPort di risposta; altrimenti, quando il packet ritorna, l'AmigaDOS ha già sovrascritto la porta originale. L'AmigaDOS mantiene tutti gli altri campi eccetto quelli del risultato.

Tutti i packet dell'AmigaDOS sono mandati alla message port creata come parte di un processo; questa message port è inizializzata in modo che l'arrivo di un messaggio causi l'impostazione a 1 del segnale (signal, in inglese) 8. Un processo dell'AmigaDOS che sta aspettando un messaggio, attende in

effetti che sia impostato a 1 il segnale 8. Quando questo avviene, il processo riprende e GetMsg accede al messaggio dalla message port estraendo l'indirizzo del packet. Se il processo è uno di quelli di gestione dell'Amiga-DOS, il packet contiene un valore nel campo PktType che indica l'azione da eseguire, come per esempio leggere alcuni dati. I campi degli argomenti contengono informazioni specifiche come l'indirizzo e la dimensione del buffer nel quale devono essere inseriti i caratteri.

Quando il processo di gestione ha completato il lavoro necessario per soddisfare la richiesta, il packet viene rimandato al mittente, usando la stessa struttura message. Sia la struttura message che la struttura packet devono essere allocate dal task e non devono essere disallocate prima che sia giunta la risposta. Solitamente l'AmigaDOS viene chiamato dal task per inviare il packet, come quando si effettua una chiamata a Read. Comunque ci sono casi nei quali è richiesto l'I/O asincrono, e il task può mandare i packet al processo di gestione quando è necessario. Le strutture message e packet devono già essere allocate e il campo ProcessID deve essere aggiornato con la message port alla quale dev'essere restituito il packet. Una chiamata a PutMsg manda il messaggio al destinatario. Occorre notare che i packet spediti possono ritornare sia alla stessa porta, sia a porte differenti.

Nella versione 1.2 c'è un nuovo tipo di packet: SetFileDate (34). Questo nuovo tipo di packet si usa per impostare la data di un file o di una directory a un valore particolare. Il primo argomento è un lock e il secondo è un APTR a una data AmigaDOS restituita dalla routine DateStamp( ).

C'è un altro nuovo tipo di packet: SetRawMode (994). Questo nuovo packet è usato per cambiare il device CON: in modo che si comporti come il device RAW:, e per riportarlo allo stato iniziale. Includendo l'argomento TRUE (vero) si seleziona il modo raw (si comporta come RAW:); includendo l'argomento FALSE (falso) si riporta il device console allo stato iniziale.

Altri packet includono Flush (27) e MoreCache (18).

C'è comunque una differenza tra il device RAW: e il device CON: funzionante in modo raw. CON:, sia in modo normale sia in modo raw, accetta le sequenze di escape per abilitare e disabilitare la conversione di un carattere di nuova linea (LF) in ritorno carrello + nuova linea (CR+LF); RAW: non converte i Line Feed (LF). Le sequenze sono <CSI>20h per abilitare la conversione e <CSI>201 per disabilitarla.

3.8.1 Tipi di packet

L'AmigaDOS supporta i packet riportati nelle pagine successive. Non tutti i tipi sono validi per tutti i processi di gestione, per esempio una richiesta rename è valida solo per gestori che supportano il filing system. Per ogni packet sono descritti gli argomenti e i risultati. Il codice decimale attuale appare vicino al nome simbolico. In tutti i casi il campo Res2 contiene informazioni aggiuntive riguardo a un errore, indicato nella maggior parte

dei casi da Res1 impostato a zero. Per ottenere questa informazione addizionale, quando si usa una funzione standard dell'AmigaDOS, si può richiamare IoErr.

Open old file

```
Tipo   LONG   Action.FindInput (1005)

Arg1   BPTR   FileHandle
Arg2   BPTR   Lock
Arg3   BSTR   Nome

Res1   LONG   Booleano
```

Cerca di aprire un file esistente per accedervi in input oppure in output (per ulteriori dettagli sull'apertura dei file per l'I/O vedere la funzione Open nel capitolo 2 del *Manuale dell'AmigaDOS per il programmatore* in questo stesso volume). Per ricevere il valore di un lock si chiama prima DeviceProc per ottenere il ProcessID del gestore, e poi IoErr che restituisce il lock desiderato. In alternativa il lock e il ProcessID possono essere ottenuti direttamente dalla struttura DevInfo. Occorre notare che il lock non si riferisce al file stesso ma alla directory che lo contiene.

Il codice che effettua la chiamata deve allocare e inizializzare FileHandle. Operazione che viene compiuta azzerando tutti i campi eccetto CharPos e BufEnd, i quali devono essere posti a -1. Il campo ProcessID all'interno di FileHandle deve essere uguale all'identificatore del processo di gestione.

Se la funzione fallisce si ottiene come risultato zero, e il campo Res2 fornisce informazioni sul motivo del fallimento. In questo caso FileHandle deve essere disallocato.

Open new file

```
Tipo   LONG   Action.FindOutput (1006)

Arg1   BPTR   FileHandle
Arg2   BPTR   Lock
Arg3   BSTR   Nome

Res1   LONG   Booleano
```

Gli argomenti sono gli stessi illustrati per la voce precedente.

Read

```
Tipo   LONG   Action.Read (82)

Arg1   BPTR   FileHandle Arg1
Arg2   APTR   Buffer
Arg3   LONG   Lunghezza

Res1   LONG   Lunghezza attuale
```

Per leggere da un handle di un file, si estrae l'identificatore di processo dal campo ProcessID dell'handle e si pone nel campo arg1 del packet, il campo arg1 derivante dall'handle. L'indirizzo del buffer e la sua lunghezza vengono posti negli altri due campi degli argomenti. Il risultato indica il numero di caratteri letti; vedere la funzione Read per ulteriori dettagli. Un errore è indicato con -1 e Res2 lo documenta con maggiori informazioni.

Write

```
Tipo   LONG   Action.Write (87)

Argl   BPTR   FileHandle Argl
Arg2   APTR   Buffer
Arg3   LONG   Lunghezza

Resl   LONG   Lunghezza attuale
```

Gli argomenti sono gli stessi di Read. Vedere la funzione Write per le informazioni sul campo del risultato.

Close

```
Tipo   LONG   Action.end (1007)

Argl   BPTR   FileHandle Argl

Resl   LONG   TRUE
```

Questo tipo di packet si usa per chiudere l'handle di un file aperto. L'identificatore di processo del gestore viene ottenuto dall'handle del file. La funzione restituisce solitamente il valore booleano TRUE (vero). Lo spazio associato all'handle del file, dopo che è stato chiuso, viene rimesso nella memoria libera.

Seek

```
Tipo   LONG   Action.Seek (1008)

Argl   BPTR   FileHandle Argl
Arg2   LONG   Posizione
Arg3   LONG   Modo

Resl   LONG   Posizione precedente
```

Questo tipo di packet corrisponde alla funzione Seek. Restituisce la posizione precedente oppure -1 se si verifica un errore. L'identificatore di processo è ottenuto dall'handle del file.

WaitChar

Tipo	LONG	Action.WaitChar (20)
Arg1	LONG	Tempo
Res1	LONG	Booleano

Questo tipo di packet dà origine alla funzione WaitForChar. Bisogna inviare il packet al processo di gestione della console, con il tempo desiderato in Arg1. Il packet ritorna quando c'è un carattere da leggere, oppure quando l'intervallo di tempo è trascorso. Se il risultato è TRUE si otterrà almeno un carattere da un successivo READ.

ExamineObject

Tipo	LONG	Action.ExamineObject (23)
Arg1	BPTR	Lock
Arg2	BPTR	FileInfoBlock
Res1	LONG	Booleano

Questo tipo di packet dà origine alla funzione Examine. Estrae l'identificatore di processo del gestore dal campo ProcessID del lock. Se il lock è zero usa il gestore dei file di default, che è mantenuto nel campo FiHand del processo. Se fallisce, il risultato è zero e le informazioni aggiuntive si trovano in Res2. Il FileInfoBlock ritorna con i campi del nome e del commento come BSTR.

ExamineNext

Tipo	LONG	Action.ExamineNext (24)
Arg1	BPTR	Lock
Arg2	BPTR	FileInfoBlock
Res1	LONG	Booleano

Questo tipo di packet dà origine alla funzione ExNext, avendo gli argomenti simili a quelli del precedente Examine. Il BSTR che rappresenta il nome del file non deve essere modificato tra una chiamata di ExamineObject e differenti chiamate di ExamineNext, visto che il nome viene usato come "ancoraggio" all'interno della directory che si sta esaminando.

DiskInfo

Tipo	LONG	Action.DiskInfo (25)
Arg1	BPTR	InfoData
Res1	LONG	TRUE

Questo packet dà origine alla funzione Info. Un corretto lock del device dovrebbe normalmente ottenere l'identificatore di processo per il gestore. Il packet può anche essere mandato al processo di gestione della console e, in questo caso, il campo Volume in InfoData contiene il puntatore alla finestra aperta in vostra vece dal gestore della console.

Parent

```
Tipo   LONG   Action.Parent (29)

Arg1   BPTR   Lock

Res1   LONG   ParentLock
```

Questo packet restituisce un lock che rappresenta il genitore del lock specificato, come viene fornito dalla funzione ParentDir. Ancora una volta deve ottenere dal lock l'identificatore di processo del gestore oppure, se il lock è zero, dal campo FiHand del processo corrente.

DeleteObject

```
Tipo   LONG   Action.DeleteObject (16)

Arg1   BPTR   Lock
Arg2   BSTR   Nome

Res1   LONG   Booleano
```

Questo tipo di packet dà origine alla funzione Delete. Deve ottenere il lock da una chiamata a IoErr() immediatamente seguente a una chiamata a DeviceProc che ha avuto successo, la quale restituisce l'identificatore di processo. Il lock attualmente si riferisce alla directory che possiede l'oggetto da cancellare, come in una richiesta Open New o Open Old.

CreateDir

```
Tipo   LONG   Action.CreateDir (22)

Arg1   BPTR   Lock
Arg2   BSTR   Nome

Res1   BPTR   Lock
```

Questo tipo di packet dà origine alla funzione CreateDir. Gli argomenti sono gli stessi di DeleteObject. Il risultato è zero oppure un lock che rappresenta la nuova directory.

LocateObject

```
Tipo   LONG   Action.LocateObject (8)

Arg1   BPTR   Lock
Arg2   BSTR   Nome
Arg3   LONG   Modo

Res1   BPTR   Lock
```

Questo tipo di packet dà origine alla funzione Lock e restituisce come risultato un lock oppure zero. Gli argomenti sono gli stessi di CreateDir, con l'aggiunta di Modo, che si riferisce al tipo di accesso (esclusivo o condiviso), come Arg3.

CopyDir

```
Tipo   LONG   Action.CopyDir (19)

Arg1   BPTR   Lock

Res1   BPTR   Lock
```

Questo tipo di packet dà origine alla funzione DupLock. Se il lock di cui si richiede la duplicazione è zero, anche il duplicato rimane zero. Altrimenti l'identificatore di processo viene estratto dal lock e questo packet viene inviato. Il risultato è un nuovo lock, oppure zero se si è verificato un errore.

FreeLock

```
Tipo   LONG   Action.FreeLock (15)

Arg1   BPTR   Lock

Res1   LONG   Booleano
```

Questo tipo di packet dà origine alla funzione UnLock. Ottiene l'identificatore di processo dal lock. Non viene intrapresa nessuna azione per disallocare un lock azzerandolo.

SetProtect

```
Tipo   LONG   Action.SetProtect (21)

Arg1   Non usato
Arg2   BPTR   Lock
Arg3   BSTR   Nome
Arg4   LONG   Maschera

Res1   LONG   Booleano
```

Questo tipo di packet dà origine alla funzione SetProtection. Lock è un lock alla directory che contiene il file ottenuto da DeviceProc come descritto

precedentemente per DeleteObject. I quattro bit meno significativi di Maschera rappresentano nell'ordine: lettura, scrittura, esecuzione e cancellazione. Il bit zero è quello per la cancellazione.

SetComment

```
Tipo   LONG    Action.SetComment (28)

Arg1   Non usato
Arg2   BPTR    Lock
Arg3   BSTR    Nome
Arg4   BSTR    Commento

Res1   LONG    Booleano
```

Questo tipo di packet dà origine alla funzione SetComment. Gli argomenti sono gli stessi di SetProtect eccetto per Arg4 che è un BSTR il quale rappresenta il commento.

RenameObject

```
Tipo   LONG    Action.RenameObject (17)

Arg1   BPTR    LockSorgente
Arg2   BSTR    NomeSorgente
Arg3   BPTR    LockDestinazione
Arg4   BSTR    NomeDestinazione

Res1   LONG    Booleano
```

Questo tipo di packet dà origine alla funzione Rename. Deve contenere un lock della directory proprietaria e un nome sia per il file sorgente sia per quello destinazione. Le directory proprietarie sono ottenute da DeviceProc come già descritto per DeleteObject.

Inhibit

```
Tipo   LONG    Action.Inhibit (31)

Arg1   LONG    Booleano

Arg2   LONG    Booleano
```

Questo tipo di packet genera un'operazione del filing system che non è disponibile come funzione dell'AmigaDOS. Il packet contiene un valore booleano il quale indica che il filing system non deve più verificare, qualsiasi nuovo disco venga inserito nel drive controllato dal processo di gestione. Se il booleano è vero, si possono cambiare i dischi senza che il processo del filing system proceda alla loro verifica (processo di convalida). Mentre gli eventi che riguardano i cambiamenti di dischi sono inibiti, il tipo del disco è

segnalato come "Not a DOS disk" (Non è un disco DOS) in modo da prevenire l'accesso al disco da parte degli altri processi.

Se il booleano è falso, il filing system ritorna al suo stato normale dopo aver verificato il disco attualmente nel drive.

Questa richiesta è utile nel caso in cui si desideri scrivere un programma come DISKCOPY e vi siano numerosi inserimenti di dischi che possono presentare una struttura completa solo parzialmente. Se si usa questo tipo di packet si evita la generazione di messaggi di errore da parte del disk validator mentre cerca di leggere settore per settore un disco non completo.

RenameDisk

Tipo	LONG	Action.RenameDisk (9)
Argl	BPTR	Nuovonome
Resl	BPTR	Booleano

Anche in questo caso, il packet genera un'operazione non disponibile normalmente tramite una funzione dell'AmigaDOS. L'unico argomento indica il nuovo nome richiesto per il disco inserito nel drive (il drive gestito dal processo del file system su cui il packet è stato inviato). Il nome del volume viene modificato sia in memoria sia su disco.

Capitolo 4

Informazioni aggiuntive sull'AmigaDOS per la programmazione avanzata

Questo capitolo descrive alcuni argomenti utili per i programmatori avanzati che desiderano creare nuovi device da aggiungere al proprio Amiga, o che vogliono far eseguire il proprio codice anche da un computer dotato di oltre 512K di memoria.

Qui di seguito ne diamo un sintetico sommario.

Descrizione degli hunk di overlay
per i programmatori che devono assemblare grandi programmi.

Programma ATOM
lavora con il nuovo formato dei file binari permettendo di modificare, in modo appropriato, i bit di caricamento. Assicura che il codice e i dati del programma che devono essere residenti nella memoria Chip (i primi 512K del sistema) siano effettivamente posti in quell'area quando sono caricati dall'AmigaDOS, in modo che il programma stesso funzioni. Altrimenti il codice del programma potrebbe non funzionare con macchine dotate di memoria aggiuntiva.

Aggiunta all'AmigaDOS di un nuovo device di tipo disco
permette al programmatore di aggiungere un disco rigido o un device di tipo disco, come una parte del filing system identificabile per il nome.

Aggiunta all'AmigaDOS di un nuovo device di tipo non disco
permette al programmatore di aggiungere all'AmigaDOS porte seriali, porte parallele, tavolette grafiche, RAM disk o altro (strutture non relative al filing system).

Uso dell'AmigaDOS senza Intuition
per i programmatori che preferiscono installare e usare un proprio programma per la gestione dello schermo, al posto di quello fornito da Intuition.

Tavola degli hunk di overlay

Quando si usano gli overlay, il linker genera principalmente un file molto grande, che contiene tutti i moduli oggetto sotto forma di hunk dotati di codice rilocabile. La tavola degli hunk di overlay contiene una struttura dati che descrive i vari hunk e le relazioni che intercorrono tra loro.

Durante la realizzazione di un programma che faccia uso di overlay, bisogna tenere ben presente come il manager di overlay (chiamato anche supervisore di overlay) gestisce l'interazione tra i vari segmenti del file. Quello che si deve fare è fondamentalmente creare un albero che rifletta le relazioni esistenti tra i vari moduli di codice che fanno parte del programma complessivo, e comunicare al linker come deve costituire questo albero.

La tavola degli hunk di overlay è generata come un insieme di voci, composte da 8 long word, ognuna delle quali descrive un particolare nodo di overlay appartenente al file generale.

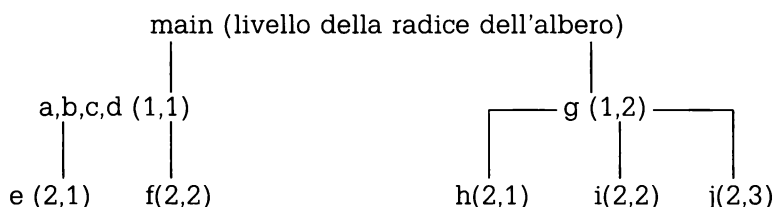
```
STRUTTURA DATI DI UNA VOCE DELLA TAVOLA
DEGLI HUNK DI OVERLAY:

long seekOffset;        /*dove si trova questo nodo nel file*/
long dummy1;            /*uno zero per compatibilità*/
long dummy2;            /*uno zero per compatibilità*/
long level;             /*livello nell'albero*/
long ordinate;          /*numero d'ordine (o ordinata) per quel
                        livello*/
long firstHunk;         /*numero di hunk del primo hunk che
                        contiene questo nodo*/
long symbolHunk;        /*numero di hunk nel quale risiede il
                        simbolo*/
long symbolOffsetX;     /*(offset+4), dove offset è l'offset
                        all'interno della tavola dei simboli
                        dell'hunk nel quale risiede il simbolo*/
```

Questi dati vengono illustrati nei paragrafi che seguono.

Progettazione di un albero di overlay

Supponiamo di avere, per esempio, i file: main, a, b, c, d, e, f, g, h, i e j. Main può richiamare a, b, c e d che a loro volta possono richiamare main. La routine e può essere richiamata da a, b, c, d oppure main, ma non è in relazione con f. Quindi, se deve essere eseguita una parte di e, anche a, b, c e d devono risiedere in memoria. La routine f è simile a e, cioè non ha bisogno della presenza di e, ma può ugualmente essere richiamata da a, b, c o d. Questo significa che il supervisore di overlay può condividere lo spazio di memoria tra e ed f, visto che nessuna delle due deve essere residente in memoria perché l'altra possa essere eseguita.

Se si considera che la routine g condivide lo spazio della combinazione a, b, c e d, e che a loro volta h, i e j condividono la stessa memoria, ecco che si avranno le basi per costruire l'albero di overlay di questo programma:

Non solo abbiamo disegnato l'albero, ma abbiamo anche etichettato i suoi rami in modo che corrispondano al numero di hunk di overlay (livello, ordinata) che si trovano nella tavola degli hunk di overlay; questi, a loro volta corrispondono ai nodi ai quali sono stati assegnati.

Dalla descrizione precedente si può vedere che, se main chiama una qualunque routine del segmento di programma a-d, tutti questi segmenti devono essere in memoria contemporaneamente. Quindi il linker li ha assegnati tutti a un solo nodo. Mentre a-d sono residenti, se si richiama una routine in e viene caricato automaticamente il modulo e, che viene inizializzato (ogni volta che viene trasferito) in modo che le routine in esso contenute siano pronte per essere eseguite. Se un qualunque segmento a-d richiama una routine in f, il linker sostituisce e con il contenuto di f, e inoltre lo inizializza. Quindi a-d si trovano al livello 1, mentre e ed f sono al livello 2, richiedendo di conseguenza che a-d siano caricati prima di e o f perché questi ultimi possano essere ricaricati e vi si possa accedere.

Nota: *una routine può effettuare chiamate solamente ad altre routine residenti in altri nodi attualmente residenti in memoria (i nodi precedenti il nodo nel quale si trova la routine attualmente in uso) o che risiedono in nodi che ne sono figli diretti. Questo significa che main non può chiamare e direttamente, mentre e può richiamare le routine contenute in main dal momento che si tratta di un nodo precedente.*

Nota: *all'interno di ogni ramo di ogni sottonodo, i numeri d'ordinata cominciano da 1 per il livello dato.*

Descrizione dell'albero

L'albero di overlay si crea comunicando al linker la sua struttura. I valori numerici, simili a quelli notati nella figura precedente, vengono assegnati sequenzialmente dal linker stesso e appaiono nella tavola dei nodi hunk. Qui di seguito si trova la sequenza di istruzioni per l'overlay che descrive la figura precedente:

```
OVERLAY
a,b,c,d
*e
*f
g
*h
*i
*j
#
```

Questa descrizione comunica al linker che a,b,c,d fanno parte di un nodo singolo collocato a un certo livello (in questo caso il livello 1); l'asterisco prima di e ed f comunica che entrambi si trovano al livello successivo ad a-d e che sono accessibili solo attraverso a-d o qualunque altro elemento che sia più vicino alla radice dell'albero. Il nome g non possiede asterisco, e quindi è considerato allo stesso livello di a-d, segnalando anche che a-d oppure g potranno essere residenti in memoria, ma non simultaneamente. I nomi h, i e j sono in relazione con g, un livello sotto di esso.

I paragrafi precedenti hanno illustrato l'origine del livello del nodo hunk e dell'ordinata hunk nella tavola dei simboli degli hunk di overlay.

Seek offset

Il primo valore di ogni nodo nella tavola di overlay è il seek offset. Come si è spiegato in precedenza, il linker, in modalità overlay, genera un unico grande file contenente tutti i nodi di overlay. Il seek offset è quel valore che deve essere fornito alla routine Seek (file, byte_offset) per puntare al primo byte dell'hunk header di un nodo.

InitialHunk

Il valore InitialHunk nella tavola dei simboli di overlay è usato dal supervisore dell'overlay quando deve rimuovere un nodo. Specifica quale hunk iniziale deve essere preventivamente caricato perché il nodo contenente questo simbolo venga a sua volta caricato. Quando si richiama

una routine a un livello e a un'ordinata differente (a meno che sia un figlio diretto, di livello successivo, del nodo corrente), diventa necessario liberare la memoria occupata da hunk non validi, per fare posto agli hunk di overlay che contengono il simbolo desiderato.

SymbolHunk e SymbolOffsetX
Queste voci sono usate dal supervisore di overlay per localizzare il punto d'entrata, una volta si sappia con certezza se è già in memoria o se deve essere ancora caricato. SymbolHunk indica in quale hunk si trova il simbolo. SymbolOffsetX-4 indica l'offset dall'inizio dell'hunk che individua l'attuale punto d'entrata.

ATOM: Modificatore del file oggetto temporaneo per Alink

Queste pagine descrivono l'utility ATOM (acronimo per Alink Temporary Object Modifier), spiegando anche la storia del suo sviluppo, il modo nel quale è stata installata e le alternative al suo impiego.

Il "problema"

I programmatori hanno bisogno di specificare quale parte dei propri programmi deve essere collocata nella memoria Chip (i primi 512K) in modo che i chip custom possano accedervi. Inoltre desiderano che questi dati siano trattati come qualunque altro dato presente nei propri programmi e di conseguenza vogliono che la fase di link e di caricamento si svolga normalmente.

Soluzioni precedenti

Prima dello sviluppo dell'utility ATOM, il modo raccomandato di affrontare il problema era di effettuare una chiamata ad AllocMem con il bit "Chip memory" impostato a 1, per poi copiare i dati dal punto in cui erano stati caricati (memoria Fast) in quello dove dovevano risiedere (memoria Chip), usando poi un puntatore per accedervi. Questa procedura implicava la presenza in memoria di due copie degli stessi dati, la prima caricata insieme al programma, la seconda copiata nei primi 512K di memoria.

L'altra "soluzione" era di avere un programma non eseguibile su macchine dotate di più di 512K. Ma questa è diventata presto una soluzione inaccettabile.

La soluzione ATOM

1. Compilare e assemblare normalmente.
2. Far passare il codice oggetto attraverso il pre-processore chiamato "ATOM". ATOM interagisce con l'utente e con i (o il) file oggetto, segnalando gli hunk desiderati "per la memoria Chip" attraverso l'alterazione del tipo dell'hunk.
3. Il linker ora riceve nove tipi di hunk invece di tre. I vecchi codici erano hunk_code, hunk_data e hunk_bss. Quelli nuovi sono:

   ```
   hunk_code_chip  = hunk_code  + bit 30 a 1
   hunk_code_fast  = hunk_code  + bit 31 a 1
   hunk_data_chip  = hunk_data  + bit 30 a 1
   hunk_data_fast  = hunk_data  + bit 31 a 1
   hunk_bss_chip   = hunk_bss   + bit 30 a 1
   hunk_bss_fast   = hunk_bss   + bit 31 a 1
   ```

   Il linker passerà tutti i tipi di hunk, concatenandoli se è necessario, al loader che userà le informazioni contenute nell'hunk header durante la fase di caricamento.
   Dalle informazioni fornite nella documentazione del linker dovreste ricordarvi che gli hunk CODE contengono linguaggio macchina eseguibile (del 68000), che gli hunk DATA contengono dati inizializzati (costanti, ...) e che quelli BSS contengono dati non inizializzati (array, dichiarazioni di variabili, ...).
4. Il loader effettuerà il caricamento seguendo le informazioni del passaggio 3, con gli hunk che verranno posti nel tipo di memoria scelto.
5. Le vecchie versioni del loader interpreteranno i nuovi tipi di hunk come hunk MOLTO grandi, e non li caricheranno generando l'errore 103: not enough memory (memoria non sufficiente).

Altre possibili soluzioni

L'assemblatore e il "C" della Lattice, o di altri produttori, possono essere modificati per generare i nuovi tipi di hunk sotto il controllo del programmatore.

Come lavorano i bit

La dimensione dell'hunk è determinata da una long word contenente il numero di long word che costituiscono l'hunk stesso. Quindi, nell'immediato futuro, anche se avremo macchine dotate di uno spazio di indirizzamento a

32 bit, i due bit più significativi rimarranno inutilizzati. Questi ultimi sono stati ridefiniti nel modo seguente:

Bit 31	Bit 30	
0	0	Se nessun bit è impostato a 1, viene usato qualunque tipo di memoria disponibile, dando la preferenza nella scelta alla memoria Fast. Questa modalità è compatibile con il vecchio approccio.
1	0	Il loader deve ottenere la memoria Fast, altrimenti fallisce.
0	1	Il loader deve ottenere la memoria Chip, altrimenti fallisce.
1	1	Se entrambi i bit sono posti a uno, allora la word successiva dispone di informazioni extra. Questa modalità è riservata per future espansioni, secondo le necessità. Attualmente non viene usata.

Il problema della compatibilità

I vecchi programmi, quelli che non sono stati compilati o assemblati con le nuove opzioni e quelli che non sono stati modificati da ATOM, non vengono eseguiti sempre nello stesso modo; se sono stati strutturati male, nella memoria estesa non vengono eseguiti del tutto. Le "soluzioni precedenti" menzionate all'inizio di questo capitolo sono ancora applicabili.

Lo sviluppo e il controllo di un programma in una macchina con 512K può seguire ESATTAMENTE lo stesso iter che si è usato finora – fase di edit, compilazione, link, esecuzione, test, fase di edit ... – FINCHÉ non si raggiunge la fase della produzione. Poi si deve usare un editor, il compilatore, ATOM, Alink, aggiungere una certa quantità di memoria esterna (più di 512K) e verificare il funzionamento. Questo procedimento funziona per qualunque ambiente di sviluppo (Amiga, IBM, Sun).

Per lo sviluppo diretto in un Amiga dotato di più di 512K si potrebbe usare ATOM con pochi file oggetto, così da poter eseguire i propri programmi in una macchina con la memoria estesa, e avere nello stesso tempo un ampio RAM disk. L'iter di sviluppo diventerebbe quindi: fase di edit, compilazione, opzionalmente un passaggio attraverso ATOM se il codice o i dati devono essere usati dal Blitter, fase di link, esecuzione, verifica, fase di edit ...

I "nuovi programmi" non vengono caricati in un ambiente dotato del Kickstart 1.0. Il risultato sarebbe un errore 103 (memoria insufficiente).

Vecchie versioni (V1.0 e precedenti) di DumpObj e OMD non lavorano con file elaborati da ATOM.

Ambiente di lavoro

Perché tutto questo funzioni, bisogna disporre di copie compatibili con la versione 1.1 di:

ATOM (versione 1.0 o successive)
Alink (versione 3.30 o successive)
Kickstart (versione 1.1 o successive) per il DOS LOADER
DumpObj (versione 1.2) necessario se si vogliono esaminare i programmi modificati con ATOM.

Sintassi della linea di comando di ATOM

La sintassi della linea di comando è:

```
ATOM <file_in> <file_out> [-I]
```

oppure

```
ATOM <file_in> <file_out> [-C[C|D|B]][-F[C|D|B]][-P[C|D|B]]
```

dove:

<file_in>	rappresenta un file oggetto, appena compilato, assemblato oppure appena passato attraverso ATOM (infatti si possono eseguire successivi passaggi attraverso ATOM).
<file_out>	la destinazione del file convertito.
-C	cambia la memoria in Chip.
-F	cambia la memoria in Fast.
-P	cambia la memoria in "Pubblica" (qualunque tipo di memoria disponibile).
C	modifica gli hunk CODE.
D	modifica gli hunk DATA.
B	modifica gli hunk BSS.

Esempi di linee di comando

Esempio 1

Nella maggior parte dei casi non c'è bisogno di allocare gli hunk CODE nella memoria Chip. Al contrario gli hunk DATA e BSS spesso hanno bisogno di essere allocati nella memoria Chip; quindi il formato che segue

rappresenta un modo consueto di usare ATOM. Perché gli hunk CODE finiscano nella RAM Pubblica e che quelli DATA e BSS finiscano nella RAM Chip bisogna usare:

```
ATOM file_in.obj file_out.obj -pc -cdb
```

Esempio 2

Perché tutti gli hunk di un file oggetto siano caricati nella memoria Chip bisogna digitare:

```
ATOM file_in.obj file_out.obj -c
```

Esempio 3

Per caricare nella memoria Chip i soli hunk DATA bisogna usare:

```
ATOM file_in.obj file_out.obj -cd
```

Esempio 4

Questo è un esempio interattivo. L'input dell'utente è in minuscolo, l'output del computer è in maiuscolo. In questo esempio l'hunk CODE è posto come "Fast" e l'hunk DATA come "Chip"; non ci sono hunk BSS. L'aiuto del programma è stato richiesto all'inizio.

```
2> atom from.o from.set -i
AMIGA OBJECT MODIFIER V1.0

UNIT NAME FROM
HUNK NAME NONE
HUNK TYPE CODE                    {Nota: hunk CODE}
MEMORY ALLOCATION PUBLIC

DISPLAY SYMBOLS [Y/N] y
_base..
_xcovf.
_CXD22..
_printf.
_main...
MEMORY TYPE? [F|C|P] ?            {Nota: richiesta di aiuto}
PLEASE ENTER   F FOR FAST      }
               C FOR CHIP      }  tipo di memoria
               P FOR PUBLIC    }
               Q TO QUIT          {cancella l'operazione senza creare alcun file
                                  di output}
```

```
              W TO WINDUP         {non modifica il resto del file}
              N FOR NEXT HUNK     {salta questo hunk, mostra il successivo}
MEMORY TYPE? [F|C|P] f

UNIT NAME 0000
HUNK NAME NONE
HUNK TYPE DATA                    {Nota: hunk DATA}
MEMORY ALLOCATION PUBLIC

DISPLAY SYMBOLS? [Y/N] n

MEMORY TYPE? [F|C|P] c

UNIT NAME 0000
HUNK NAME NONE
HUNK TYPE BSS
MEMORY ALLOCATION PUBLIC

DISPLAY SYMBOLS? [Y/N] y

MEMORY TYPE? [F|C|P] p
2>_
```

Messaggi d'errore

Error Bad Args:

a) Un'opzione non inizia con un "-".
b) Numero di parametri sbagliato.
c) "-" non seguito da I, C, F o P.
d) -x fornito in aggiunta a -I, ecc.

Error Bad infile:

Il file di input non è stato trovato.

Error Bad outfile:

Il file di output non può essere creato.

Error Bad Type ##:

ATOM ha trovato un tipo di hunk che non riconosce. Il file oggetto potrebbe essere difettoso.

Error empty input:

Il file di input non contiene nessun dato.

Error Read Externals:

Riferimento esterno o definizione di un tipo non definito. Il file oggetto potrebbe essere difettoso.

Error premature end of file:

ATOM ha incontrato una condizione di fine-file mentre stava aspettando ancora dell'input. Il file potrebbe essere difettoso.

Error This utility can only be used on files that have NOT been passed through ALINK:

Il file di input specificato è già stato elaborato dal linker. I simboli esterni sono stati rimossi e gli hunk sono stati concatenati. ATOM deve essere usato con i file oggetto prodotti dal compilatore C o dal Macro Assembler PRIMA che entrino nella fase di link.

Creazione di un nuovo device per l'AmigaDOS

Queste pagine forniscono le informazioni che riguardano l'aggiunta di nuovi device non appartenenti al filing system. A pagina 352 vengono fornite le informazioni riguardanti l'aggiunta di device relativi al filing system (hard disk, disk drive), cioè device che il DOS può usare per leggere e scrivere file e directory.

Queste informazioni si possono usare per aggiungere ad esempio una nuova porta parallela o una nuova porta seriale. Quindi si potrebbe creare un device chiamato "SER2:" che, per quanto riguarda il DOS, agisca nello stesso modo di "SER:"

Per creare un nuovo device per l'AmigaDOS bisogna compiere due passi distinti. Per prima cosa si deve creare un device appropriato, un processo che qui non è spiegato.

Nota: *in Programmare l'Amiga Vol. I è contenuto il codice per la struttura portante di un device residente su disco.*

In seguito, bisogna rendere questo nuovo device disponibile come device dell'AmigaDOS. Per far questo si deve scrivere un gestore appropriato del device (riferirsi a *Programmare l'Amiga Vol. I*) e installarlo nelle strutture dell'AmigaDOS.

Per operare questa installazione si deve creare, per il nuovo device, una struttura device-node. L'operazione è simile alla creazione di una voce DevInfo per un nuovo device disco, se si eccettua il fatto che l'argomento Startup può avere un valore qualsiasi. La voce Segment list è zero e il nome del file contenente il gestore del device residente su disco è collocato nella voce Filename.

0	Next
0	dt_device
0	Task (oppure l'identificatore di processo - vedere oltre)
0	Lock
BSTR	Nome del file contenente il codice del processo di gestione
NNN	Dimensione richiesta dello stack
NN	Priorità richiesta
XXX	Informazioni di startup
0	SegList (diverso da zero se si carica il codice)
0	Global vector richiesto
BSTR	Nome del device

Il gestore del device, che normalmente è scritto in BCPL, è l'interfaccia tra il vostro device e un programma applicativo. Il Kernel dell'AmigaDOS cerca di caricare il codice del gestore e di creare per lui un nuovo processo quando ci si riferisce a questo codice per la prima volta. Il procedimento è automatico quando l'AmigaDOS nota che il campo Task della struttura DevInfo è zero. Se il codice è già stato caricato, il puntatore al segmento viene posto nel campo SegList. Se questo campo è zero l'AmigaDOS carica il codice contenuto nel file il cui nome è indicato dal campo Filename, e aggiorna il campo SegList.

Se si desidera che il caricamento automatico e il processo di inizializzazione funzionino, bisogna creare un modulo di codice che sia scritto in BCPL, oppure in Assembly in modo che assomigli a un modulo BCPL. Questo assicura che il collegamento dinamico usato dal nucleo centrale dell'AmigaDOS funzioni correttamente.

Se si sta scrivendo in Assembly, il formato della sezione di codice deve essere quello illustrato qui di seguito. Si possono usare anche le sezioni DATA e BSS, ma ogni sezione deve avere lo stesso formato.

```
StartModule   DC.L   (EndModule-StartModule)/4    Dimensione del
                                                  modulo in long word
EntryPoint    ....
              ....   (il vostro codice)
              CNOP   0,4                          Allineamento
              DC.L   0                            Segnalatore di fine
              DC.L   1                            Definisce 1 globale
              DC.L   EntryPoint-StartModule       Offset del punto d'inizio
              DC.L   1                            Globale più alto usato
EndModule     END
```

In Assembly si inizia con il registro D1, che mantiene un puntatore BCPL al packet iniziale restituito dal Kernel.

Se si sta scrivendo in BCPL, la struttura portante della routine apparirà come segue. Il lavoro principale del processo di gestione del device è di convertire le richieste Open, Read, Write e Close nelle richieste read e write per il device. Gli altri tipi di packet sono segnalati con un errore.

"I file include contengono costanti utili"

```
GET ''LIBHDR''
GET ''IOHDR''
GET ''MANHDR''
GET ''EXECHDR''
```

Questo è un gestore per la struttura di un task.

Quando il processo viene creato, il packet di parametri contiene quanto segue:

parm.pkt!pkt.argl = BPTR a una stringa BCPL del nome del device (per esempio "SKEL:")
parm.pkt!pkt.arg2 = informazioni extra (se necessarie)
parm.pkt!pkt.arg3 = BPTR al nodo info device

```
MANIFEST
$(
IO.blocksize = 30     (Dimensione dei blocchi IO dei device)
$)

LET start (parm.pkt) BE
$(
LET extrainfo    =  parm.pkt!pkt.arg2
LET read.pkt     =  0
LET write.pkt    =  0
LET openstring   =  parm.pkt!pkt.argl
LET inpkt        =  VEC pkt.resl
LET outpkt       =  VEC pkt.resl
LET IOB          =  VEC IO.blocksize
LET IOBO         =  VEC IO.blocksize
LET error        =  FALSE
LET devname      =  ''serial.device*X00''
LET open         =  FALSE (Flag per sapere se il device è stato "aperto"
                    con act.findinput o act.findoutput)
LET node         =  parm.pkt!pkt.arg3
```

(Azzera il blocco prima di compiere altre operazioni, in modo che si possa sapere cosa viene scritto quando si richiama OpenDevice)

```
FOR i=0 TO IO.blocksize DO IOB!i := 0
IF OpenDevice ( IOB, devname, 0, 0 ) = 0 THEN error := TRUE

IF error THEN
$( returnpkt (parm.pkt,FALSE,error,objectinuse)
  return
$)
```

(Copia tutti gli info necessari nel buffer di output)

```
FOR i=0 TO IO.blocksize DO IOBO!i := IOB!i

outpkt!pkt.type := act.write
inpkt!pkt.type := act.read
```

```
node!dev.task := taskid()        (Inserisci l'identificatore di processo nel
                                 nodo device)
```

(Il lavoro con il packet dei parametri è terminato... restituisci il packet...)

```
returnpkt(parm.pkt,TRUE)
```

(Questo è il loop principale che attende un evento)

```
$( LET p = taskwait()
SWITCHON p!pkt.type INTO
$(
CASE act.findinput:        (Open)
CASE act.findoutput:
$( LET scb = p!pkt.argl
    open := TRUE
    scb!scb.id := TRUE     (Interattivo)
    returnpkt(p,TRUE)
    LOOP
$)
CASE act.end:              (Close)
    node!dev.task :=0      (Rimuovi l'identificatore di processo dal nodo del
                           device)
    open := FALSE
    returnpkt(p,TRUE)
    LOOP

CASE act.read:             (Restituisce la richiesta Read)
    inpkt := p
    handle.return(IOBO,read.pkt)
    LOOP

CASE act.write:            (Restituisce la richiesta Write)
    outpkt := p
    handle.return(IOBO,write.pkt)
    LOOP

CASE 'R':                  (Richiesta di lettura)
    read.pkt := p
    handle.request(IOB,IOC.read,p,inpkt)
    inpkt := 0
    LOOP

CASE 'W':                  (Richiesta di scrittura)
    write.pkt := p
```

```
handle.request(IOBO,IOC.write,p,outpkt)
outpkt := 0
LOOP
 DEFAULT:
UNLESS open DO node!dev.task := 0 (Rimuovi l'identificatore di processo a
                                   meno che non sia aperto)
 $)
$) REPEATWHILE open|outpkt = 0|inpkt = 0
Termination
CloseDevice( IOB )
```

(Gestisci una richiesta IO. Comando passato, packet di trasmissione (tp) e packet di richiesta (rp). rp contiene il buffer e la lungheza in arg2/3).

```
AND handle.request(IOB,command,rp,tp) BE
LET buff = rp!pkt.arg2
LET len = rp!pkt.arg3
SetIO( IOB, command, ?, rp!pkt.arg3, 0 )
putlong( IOB, IO.data, buff )
SendIO(IOB,tp)
```

(Gestisci il ritorno di una richiesta di IO. Il packet di richiesta dell'utente è passato come p e deve essere restituito con messaggio di successo/fallimento).

```
AND handle.return(IOB,p) BE
$(
LET errcode = IOB 0.error
LET len = getlong(IOB,IO.actual)
TEST errcose = 0 THEN                      (Nessun errore)
  returnpkt(p,len)
ELSE
  returnpkt(p,-1,errcode)
```

Se si desidera scrivere il proprio gestore del device in C, non si possono utilizzare il caricamento automatico e la creazione del processo forniti dal Kernel dell'AmigaDOS. Quindi si deve caricare il codice per conto proprio e usare una chiamata a CreateProc per creare il processo. Il risultato di questa chiamata dev'essere posto nel campo Task della struttura DevInfo. Poi si deve inviare un messaggio, che contenga informazioni quali il numero d'unità del device in questione, affinché il processo inizi a essere eseguito. Il processo di gestione si pone in attesa delle chiamate Open, Read, Write e Close per elaborarle come descritto nell'esempio precedente. Il codice C non ha bisogno di inserire l'identificatore di processo nel nodo del device, poiché questo viene fatto quando il codice viene caricato, come descritto in precedenza.

Creazione di nuovi device disco

Per creare un nuovo device disco, si deve costituire un nuovo nodo di device come descritto nella sezione 3.3.1 di questo manuale. Si deve inoltre scrivere un gestore di device relativo al nuovo device disco.

Un gestore di device per un nuovo device disco deve emulare tutte le chiamate che sono eseguite dal device trackdisk (descritto in *Programmare l'Amiga Vol. I*). Deve essere in grado di rispondere a comandi tipo Read, Write, Seek e restituire informazioni sullo stato del dispositivo, nella stessa maniera descritta per il gestore del trackdisk.

Nella descrizione seguente, la maggior parte dei puntatori sono di tipo BPTR (già descritto precedentemente a pagina 318), un puntatore a una locazione di memoria allineata secondo una long word (come restituito da AllocMem) con uno scorrimento verso destra di due posti.

Si costruisca il nuovo nodo con i campi seguenti:

0	Next
0	dt_device
0	Task
0	Lock
0	Handler
210	Stacksize
10	Priority
BPTR	Puntatore all'info di startup
Seglist	
0	Global vector
BSTR	Puntatore al nome

Il BSTR è un puntatore BCPL al nome del nuovo device (come HD0:), rappresentato dalla lunghezza della stringa nel primo byte, seguito dai caratteri.

Seglist deve essere la lista di segmenti del task del filing system. Per ottenere questa informazione si deve accedere a un campo nella struttura base di processo di uno dei task del filing system.

Il codice seguente può essere usato per questo scopo.

```
   UBYTE *port;
port  =  DeviceProc( ''DF0:'');                    /* Retituisce la porta msg del task
                                                   del file system */
task  =  (struct Task *) (port-sizeof(struct Task); /* La struttura task e' dopo la
                                                   porta*/
list  =  ( task.pr_Seglist );                      /* Costruisce un ptr da
                                                   SegArray */
```

```
segl  =  list[3];                /*Il terzo elemento nel SegArray e' la seglist
                                 del file system */
```

In seguito, bisogna impostare le informazioni di startup (anche qui bisogna ricordarsi di usare i BPTR dove è necessario). Queste informazioni consistono in un BPTR a tre long word che contengono:

- Numero d'unità (non usare unità zero)
- Nome del gestore del device, memorizzato come BPTR al nome del gestore del device che deve essere seguito da un byte nullo incluso nel conto (per esempio, 4/'H'/'D'/'0'/0).
- BPTR alle informazioni del disco che contengono i seguenti campi di long word:

11	Dimensione della tavola
128	Dimensione di un blocco di disco in long word (assumendo che sia di 512 byte)
0	Settore di origine (il primo settore è il settore zero)
Numero di facce	(2 per i floppy disk)
1	Numero di settori per blocco
Numero di blocchi per traccia	(11 per i floppy disk)
2	(o più, indicanti il numero di blocchi da riservare all'inizio)
0	Fattore di pre-allocazione
0	Fattore di interleave
Numero del cilindro più basso	(solitamente 0)
Numero del cilindro più alto	(79 per i floppy disk)
5	(o più, indicanti il numero di blocchi cache)

Infine il nodo del device deve essere unito al termine della lista dei nodi di device (notate che tutti i campi Next sono BPTR) all'interno della sottostruttura Info.

Per suddividere in partizioni un disco rigido, si creano due o più nodi di device, impostando il numero del cilindro più basso e di quello più alto, in modo da eseguire l'operazione come meglio si desidera.

Uso dell'AmigaDOS senza Workbench/Intuition

Quanto segue intende dare ai programmatori alcune notizie su come l'AmigaDOS e Intuition interagiscono tra loro. È sconsigliabile disattivare interamente Intuition o l'input device. Per filtrare l'input è possibile installare il proprio processo di gestione dell'input all'interno del flusso di input (come viene dimostrato in *Programmare l'Amiga Vol. II*, descrizione dell'input device) e quindi gestire gli eventi di input autonomamente, dopo che il proprio programma è stato caricato ed eseguito dall'Amiga-DOS. Se, successivamente, si riesce a prendere il controllo della macchina in qualche modo, si può evitare che l'AmigaDOS cerchi di visualizzare i requester di sistema o, al contrario, di interagire con lo schermo modificando il DOS come mostreremo in seguito. Principalmente, il programma dovrà essere in grado di fornire un modo alternativo di gestire gli errori che causano la visualizzazione, da parte del DOS, di un requester.

Un'altra tecnica per prendere il controllo della macchina, è di ignorare del tutto il filing system dell'AmigaDOS e usare il trackdisk.device per caricare il proprio codice e i propri dati in maniera indipendente. Per realizzare questa scelta, si possono trovare i particolari riguardanti il Boot block del disco in *Programmare l'Amiga Vol. II* e la formattazione delle tracce nell'*Amiga ROM Kernel Manual*.

Illustriamo ora i particolari che riguardano l'AmigaDOS e Intuition.

L'AmigaDOS inizializza se stesso e apre Intuition. Cerca poi di aprire il file di configurazione (creato da Preferences) e lo passa a Intuition. Successivamente attiva una finestra CLI iniziale tramite Intuition e cerca di eseguire il primo comando CLI. Questo è di solito un LOADWB (carica il Workbench), seguito da un ENDCLI per il CLI iniziale.

Un programma applicativo può essere creato in modo che si comporti come il Workbench, nel senso che può generare nuovi processi. Il successivo comando CLI è ENDCLI, il quale provvede a chiudere qualunque cosa, lasciando attivo solo il nuovo processo (insieme ai processi del file system). Quest'ultimo dovrebbe impostare il campo pr_WindowPtr a -1, indicando che il DOS deve riportare gli errori in modo quiet (senza segnalazioni). Bisogna notare che l'applicazione deve provvedere alla gestione di tutti gli errori; il capitolo 3 fornisce ulteriori informazioni su questo argomento. Il DOS avrà anche inizializzato il campo TrapHandler del task dell'utente, in modo che punti al codice che mostra un requester dopo un errore; questo valore deve essere sostituito per poter adoperare una routine fornita dall'utente. Tutto ciò evita che Intuition

sia usato da parte del processo dell'utente, sempre che non ci siano gravi problemi nella gestione della memoria con perdita di dati, nel qual caso il DOS richiamerebbe direttamente la funzione Alert dell'Exec.

C'è ancora il problema che il processo del file system possa richiedere un requester nel caso di un errore nel disco o nel caso che esso stesso vada in "crash" a causa di problemi con la memoria. Per evitare che questo accada, i campi pr_WindowPtr e tc_TrapHandler del processo del file system devono essere posti a -1 e un gestore di Trap privato deve essere fornito nello stesso modo usato per il task dell'utente.

Per farlo, bisogna trovare la message port di ogni task del file system richiamando DeviceProc() e passando df0:, df1: ecc. come parametri. Un errore indica che il device non è presente. Dalla message port si può ricavare la base di ogni task del file system, e quindi si possono modificare i due campi cercati, avendo l'accortezza di ripetere l'operazione per ogni unità disco installata.

Il programma applicativo potrebbe ora chiudere Intuition. Naturalmente il Workbench non dev'essere mai stato caricato. Bisogna notare che al momento in cui scriviamo non è possibile evitare che il DOS apra Intuition.

Se l'applicazione che si desidera usare apre qualche altro device, come SER:, il processo di gestione deve essere modificato nello stesso modo usato per i processi del file system. L'applicazione non deve naturalmente aprire CON: o RAW:, una volta che Intuition è stato disattivato.

Indice analitico

A

ADDBUFFERS (comando per l'utente), 48
ALINK (comando per il programmatore), 121, 212, 257, 259
Area di lavoro non inizializzata, 303
Argomenti, 5, 65, 160, 194, 195
ASSEM (comando per il programmatore), 122, 233
Assemblatore, assembler, 235
Assembly (linguaggio), 202
ASSIGN (comando per l'utente), 39, 49
Assoluto (simbolo), 241
ATOM (Alink Temporary Object Modifier), 341

B

BINDDRIVERS (comando per l'utente), 50
Blocco di break, 314
Blocco di debug, 310
Blocco radice (root block), 289
Boot, 43
BREAK (comando per l'utente), 51

C

C, ambiente iniziale, 203
Campo del codice dell'istruzione, 238
Campo del commento, 239
Campo dell'operando, 239
Cancellazione di linee, 155, 167
Cancellazione di testo, 134
Caratteri bracket, 75
Caricamento di codice, 230, 231
CD (comando per l'utente), 32, 52
CHANGETASKPRI (comando per l'utente), 53
Chiamate all'AmigaDOS, 213, 214
CLI - vedere Comando, interfaccia linea
Close (funzione), 215
Close (packet), 331
Coda di output, 194
Codice oggetto, 43
Codifica del programma, 236
Comando/i,
  corrente di alterazione stringa, 195
  definizione,195
  d'uso più comune, 23
  esecuzione, 4, 18, 19, 70, 108, 229
  estesi, 131, 135, 142, 144
  file di, 19, 177
  formato, 20
  immediati, 131, 132, 143
  in background, 18, 108
  input e output del, 19, 33
  interfaccia linea, 24, 40, 41, 69, 98, 117, 128, 195, 202, 319
  linea di, 257
  per il programmatore, 121, 129
  per l'utente, 43, 127
  riconoscimento dei, 5
  ripetizione, 135, 142, 162
  sequenza di, 112, 128
  sintassi, 159

struttura dei file, 76
template, 196, 20
uso, 18
Combinazione con il control, 195
Commenti, 237, 239
Condizioni, 85
Contatore di programma (PC), 234
Controllo del blocco, 138
Controllo del cursore, 132
Controllo del programma, 136
COPY (comando per l'utente), 35, 54
CopyDir (packet), 334
CreateDir (funzione), 215, 333
CreateDir (packet), 333
CreateProc (funzione), 227, 320
Cross development, 205
CTRL-X (combinazione control), 4
CurrentDir (funzione), 216

D

Data block, 295, 303
Data, 33, 56, 228
DATE (comando per l'utente), 33, 56
DateStamp (funzione), 228
Default, 44
Delay (funzione), 228
DELETE (comando per l'utente), 34, 58
DeleteFile (funzione), 216
DeleteObject (packet), 333
Delimitatori, 195
Device corrente, 9
Device, creazione di, 352-355
Device logici, 14, 44, 49, 324
DeviceProc (funzione), 228
DIR (comando per l'utente), 29, 37, 59
Diramazioni, 234
Directory, 6, 29, 49, 52, 59, 91, 195
Directory,
blocco della, 291
cambiamento di nome, 107, 223
cancellazione, 216
convenzioni, 14
corrente, 8, 29, 32, 195, 216
creazione, 36, 215
esame delle, 217
genitore, 222
lock, 220, 225
protezione, 225
radice, 7, 195
Direttiva OVERLAY, 263
Direttive, 244
Direttive per il controllo dell'assemblaggio, 246
Direttive per il controllo del list, 250
Direttive per la definizione dei dati, 249
Direttive per la definizione dei simboli, 248
Direttive per l'assemblaggio condizionato, 252
Direttive per le macro, 253
Disco/dischi,
cambiamento di nome, 29, 106
copia, 27, 62
floppy, 83
formattazione, 27, 83
informazioni sul, 219
installazione, 28, 88
sistema, 44
DISKCHANGE (comando per l'utente), 61
DISKCOPY (comando per l'utente), 27, 62
DISKDOCTOR (comando per l'utente), 64
DISKED (editor per i dischi), 296
DiskInfo (packet), 332
DISKINSERTED, 63
Divisione e unione di linee, 174
DOWNLOAD (comando per il programmatore), 124
Drive corrente, 9, 195
Duplicazione di lock, 216
DupLock (funzione), 216, 334

E

ECHO (comando per l'utente), 65
ED (comando per l'utente), 66, 130
EDIT (comando per l'utente), 67, 146
Editing - vedere Editor di linea, Editor di schermo
Editor di linea, 67, 146
Editor di schermo, 66, 130
ENDCLI (comando per l'utente), 41, 69
Errori, 80, 105, 120, 189, 203, 219, 262, 268
Espressioni, 239
Etichette (label), 90, 237, 238
Examine (funzione), 217, 332
ExamineNext (packet), 332
ExamineObject (packet), 332
EXECUTE (comando per l'utente), 70
Execute (funzione), 229
Exit (funzione), 229, 321
ExNext (funzione), 217, 332

F

FAILAT (comando per l'utente), 80
FAULT (comando per l'utente), 81
File,
- apertura, 221
- batch iterativo, 79
- blocco di intestazione, 293
- cancellazione, 34, 58, 216
- caricabile, 257, 299, 310
- concatenare i, 89
- copia, 35, 54
- definizione, 195
- destinazione, 195
- di input, 178, 219
- di output, 179, 221
- di parametri, 258
- di testo, 33, 66, 67
- directory genitore, 222
- esame del, 217
- funzioni per il trattamento, 215
- handle, 44, 219, 221, 326, 330, 331
- list block, 294
- localizzazione, 37
- lock, 220, 225
- nomi di, 6, 34, 91, 107, 195, 223
- oggetto, 257, 299, 301
- protezione, 225
- raggruppamento, 121
- sequenziale, 195
- simulazione della copia di, 76
- sorgente, 195
- struttura dei, 289, 298
- system, 5, 31, 87, 289
- trattamento del contenuto, 222, 226
- utility per, 127
- WITH, 260

FILENOTE (comando per l'utente), 82
Filing system. Vedere File, system
Floppy disk. Vedere Disco, floppy
FORMAT (comando per l'utente), 27, 83
FreeLock (packet), 334
FROM, 99
Funzioni, 214

G

Gestione della fine-del-file (EOF), 165
Gestione della memoria, 128
Gestione del sistema, 128
Gestore della console. Vedere Gestore del terminale
Gestore del terminale, 4, 195
Gruppi di comandi, 161

H

Hunk, 300, 301-314

I

IF (comando per l'utente), 85
Indirizzo, 234
INFO (comando per l'utente), 31, 87

Info (funzione), 219, 332
Inhibit (packet), 335
INITIALIZE (comando per l'utente), 84
Input (funzione), 219
Input binario primario, 258
Input con la tastiera, 271, 276
Input della console, 269
Input, file di. Vedere File di input
Inserimento di linee, 155, 167
Inserimento di testo, 132, 184
INSTALL (comando per l'utente), 28, 88
Interruzione dell'AmigaDOS, 20
IoErr (funzione), 219
IsInteractive (funzione), 220
Istruzioni, 234, 237, 244
Iterazioni, 180

J

JOIN (comando per l'utente), 89

K

Keyword, 20, 21, 195

L

LAB (comando per l'utente), 90
Libreria associata, 258, 300
Librerie, 201, 213, 258, 300, 312, 322
Librerie residenti, 201, 213, 258, 300, 311
Libs (directory), 63
Linea corrente, 150, 153, 163, 166, 171, 183, 195
Linguaggio assembly, 122
Linguaggio assembly MC68000, 122
Linker, 121, 212, 257, 314
LIST (comando per l'utente), 30, 91
LIST (direttiva), 250
Liste di segmenti, 326
Literal ASCII, 242
Loader, 314, 317
LoadSeg (funzione), 230
LocateObject (packet), 334
Locazione zero, 262
Lock, 318, 327
Lock (funzione), 220
Long word, 234

M

Macro Assembler, 233
Maiuscolo, 214
MAKEDIR (comando per l'utente), 36, 95
Memoria, 196, 230, 231, 303, 325
Microprocessore 68000, 233
Modi di indirizzamento, 242
Modifica del testo, 141
Modo esteso, 131, 196
Modo immediato, 131, 196
MOUNT (comando per l'utente), 96
MS-DOS, 211
Multitasking, 3, 196

N

NEWCLI (comando per l'utente), 40, 98
Nodo, 300, 313
NOICONS, 83
Nome di volume, 10, 44, 106, 196
Nomi dei comandi, 159
Nomi di device, 11, 44, 49, 196, 324
NOTEPAD (comando per l'utente), 100
Numeri, 241
Numeri binari, 242
Numeri decimali, 242
Numeri di linea, 150, 151, 161, 163, 185
Numeri esadecimali, 242
Numeri ottali, 242

O

Open (funzione), 221
Open New File (packet), 330
Open Old File (packet), 330
Operatori, 239

Operatori, operandi relativi agli, 240
Operazioni globali, 180
Ora, 33, 56, 110, 228
Output (funzione), 221
Output da console, 269
Output di schermo, 272, 273
Output MAP, 262
Output XREF, 262
Output, file di. Vedere File di output
Overlay,
  blocco con la tavola, 313
  file di, 258
  gestione, 263
  nodi, 300, 313
  numero di, 267
  riferimenti in, 267
  supervisore, 257
  tavola degli hunk di, 338

P

Packet, 328
Parametri di default, 74
Parametri, file di. Vedere File di parametri
Parent (packet), 333
ParentDir (funzione), 222, 333
PATH (comando per l'utente), 101
Percorsi di ricerca, 38
Porta parallela, 126
Porta seriale, 126
Porzioni di linea, 169, 196
Posizione corrente del cursore, 139, 195
Posizione logica, 223
Priorità, 4, 196
Processo, 4, 196, 227, 319
Processo di convalida, 23
Processo di gestione, 318
Programmazione, 199
PROMPT (comando per l'utente), 103
Prompt, 103, 162
PROTECT (comando per l'utente), 31, 104
Puntatore, 196
Puntatore al carattere, 172
Puntatore alla base della libreria, 322

Q

Qualificatori, 152, 196
QUIT (comando per l'utente), 105

R

RAM (device), 12
READ (comando per il programmatore), 126
Read (funzione), 222, 330
Read (packet), 330
Reboot, 44
Registri dei dati, 234
Registri di indirizzo, 234
Registro (simbolo), 241
Registro di stato, 235
RELABEL (comando per l'utente), 29, 106
Relativo (simbolo), 241
RENAME (comando per l'utente), 34, 107
Rename (funzione), 223
RenameDisk (packet), 336
RenameObject (packet), 335
Ricerca, 101, 109, 140, 161
Ridirezione, 19, 33, 46
Riferimenti esterni, 299
Rilocazione, 304-306
Root Block, 289
RUN (comando per l'utente), 108

S

Salti, 234
Scrolling, 135
SEARCH (comando per l'utente), 109
Seek (funzione), 223, 331
Seek (packet), 331
SetComment (funzione), 224, 335

SetComment (packet), 335
SETDATE (comando per l'utente), 110
SETMAP (comando per l'utente), 111
SetProtect (packet), 334
SetProtection (funzione), 225, 334
Simboli, 240, 255, 266, 309, 314
Simboli esterni, 255, 307, 314
Sintassi, 159, 196, 213
SKIP (comando per l'utente), 112
SORT (comando per l'utente), 114
Sostituzione, 140
Sostituzione di parametri, 73
Sottostruttura Info, 323
Spazi di trailing, 184, 196
STACK (comando per l'utente), 116
Startup-Sequence (file execute), 38
STATUS (comando per l'utente), 117
Storage
Stream, 44
Stringa di caratteri, 196
Stringa di testo, 109
Stringa nulla, 172
Stringhe, 160, 161, 164, 171
Stringhe multiple, 160
Stringhe qualificate, 153, 161, 164, 196
Strutture dati, 318
Struttura dati globale, 322
Struttura dei file binari, 298
Sun (computer), 205
Sviluppo di programmi, 200

T

Template, 20, 196
Terminale virtuale, 220
Termine del programma, 203
Trasferimento dei programmi, 124
Trattamento dell'output, 165
TYPE (comando per l'utente), 33, 118

U

Unità di programma, 299
UnLoadSeg (funzione), 231
UnLock (funzione), 225, 334

V

Valori, 213, 214
Valori booleani di ritorno, 214
Valori degli switch, 161
Valori dei registri, 213
Variazione di indirizzo, 244
Variazione di memoria, 244

W

WAIT (comando per l'utente), 119
WaitChar (packet), 332
WaitForChar (funzione), 226, 332
WHY (comando per l'utente), 120
Wild card (jolly), 6, 196
Word, 234
Word di operandi, 234
Word di operazione, 234
Workbench, 24, 204
Write (funzione), 226, 331
Write (packet), 331

IL MANUALE DELL'AMIGADOS

Commodore-Amiga Inc.

Questo libro è *l'unica documentazione ufficiale* della Commodore sull'AmigaDOS, il sistema operativo multitasking dei computer Amiga 500/1000/2000. È una guida indispensabile per gli utenti che de iderano sfruttare a fondo le risorse offerte dai computer della serie Amiga. Il volume si divide in tre sezioni principali:

Il Manuale per l'utente, che descrive tutti i comandi dell'AmigaDOS impartibili tramite l'interfaccia linea comando CLI. L'insieme di questi comandi consente di eseguire complesse operazioni di gestione del sistema, anche in multitasking.
Il Manuale per il programmatore, che costituisce un'utile fonte di informazioni per creare applicazioni nei linguaggi C e Assembly. Vengono descritti i possibili ambienti di sviluppo delle applicazioni, le chiamate alle routine dell'AmigaDOS contenute nella libreria residente, il compilatore Macro Assembler e il linker.
Il Manuale di riferimento tecnico, che illustra l'organizzazione su disco dei dati, dei file oggetto e dei file caricabili gestiti dall'Amiga, e le strutture dei dati impiegate dall'AmigaDOS.

Questo volume è stato scritto dagli stessi tecnici che hanno progettato e realizzato l'Amiga ed è quindi la guida più completa e affidabile sul DOS di questo elaboratore. Sia i nuovi utenti sia i programmatori vi troveranno tutte le informazioni necessarie per sfruttare nel modo migliore le risorse che l'AmigaDOS mette a disposizione.

In quest'opera sono inclusi tutti i comandi del DOS. Scopriteli e fate conoscenza con:

- AmigaDOS – Analisi del sistema operativo multitasking
- CLI – L'interfaccia linea comando per interagire con l'AmigaDOS
- I comandi DOS – Alla scoperta di tutti i comandi disponibili
- Lo screen editor – Creazione e modifica dei file di testo con ED
- Il line editor – Elaborazione sequenziale dei file di testo con EDIT
- Programmazione – Uso del C e dell'Assembly in ambiente AmigaDOS
- Chiamate all'AmigaDOS – Uso della libreria residente
- L'I/O da console – Descrizione del terminale virtuale dell'Amiga
- Il filing system – Strutture dei dati su disco e loro recupero
- Strutture dell'AmigaDOS – Strutture dei dati impiegate dall'AmigaDOS
- La programmazione avanzata – Struttura overlay dei file
- Aggiunta di nuovi device – Specifiche per l'aggiunta di device al filing system

ISBN 88-7803-002-3
9 788878 030022

Lire 60.000